National Library of Canada
Canadian Theses Service
Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Service des thèses canadiennes

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

NL-339 (r. 88/04)

Canada

THE UNIVERSITY OF ALBERTA

AN INTERACTIVE USER INTERFACE FOR A SEMICONDUCTOR DEVICE SIMULATION PACKAGE.

BY
(C)
RASHAD M. SAAD

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND
RESEARCH IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF SCIENCE.

DEPARTMENT OF ELECTRICAL ENGINEERING.

EDMONTON, ALBERTA.
(SPRING 1988)

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-42915-1

THE UNIVERSITY OF ALBERTA
RELEASE FORM

NAME OF AUTHOR : RASHAD M. SAAD

TITLE OF THESIS : AN INTERACTIVE USER INTERFACE FOR A SEMICONDUCTOR DEVICE SIMULATION PACKAGE.

DEGREE : MASTER OF SCIENCE

YEAR THIS DEGREE GRANTED : 1988

Permission is hereby granted to THE UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

5007 - 106 st.
Edmonton, Alberta
Canada T6H 2S8

Date : April 27th, 1988.

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommended to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled An Interactive User Interface for a Semiconductor Device Simulation Package submitted by Rashad Mohamed Saad, in partial fulfillment of the requirements for the degree of Master of Science.

H P Baltes

Supervisors

W. Allegretto

Date : April 19, 1988.

To my family
for all that they have done.

ABSTRACT

The simulation of a semiconductor device is concerned with the study of the behavior of the device charge carriers under the influence of electromagnetic fields. Such behavior is modeled by a set of coupled partial differential equations. A semiconductor device simulator provides a user with means to solve these models. Due to the nature of the equations their solution in closed form is impossible. As a result, the models are approximated to a set of non-linear algebraic equations. The solution of these equations require the discretization of the device in space domain. Accordingly, the solution steps are divided into three main steps. The first step is device description. In this step the device geometry is defined and certain parameters are assigned. These parameters pertain to the semiconductor device model and are known as the *physical parameters*. The second step is the discretization of the device, the result is a mesh in space domain. The final step is solving the non-linear algebraic equations per each mesh point. The result of the third step is the distribution of the electrostatic potential within the device.

This work provides the simulator user with tools and mechanisms to implement the first step of the simulation sequence. The geometric and physical description of the device are stored in dynamic and versatile tree structures. The trees are manipulated by a set of well defined operations. Such operations provide easy and consistent means of creation, modification and deletion of the device description. The description tools are offered to the user through a novel implementation of the state of the art user interface techniques. This interface is of a *graphical nature* where the description, discretization and solution steps are offered to the user through a set of *panels*. Activating a panel results in either providing the user with a hierarchy of *menus* or invoking a particular simulation step. The menus are groups of geometric forms and physical models through which a user can describe the simulated device. The user interface concurrently invokes simulation steps while assuming the responsibility of data monitoring. The user is informed in case of errors or data gaps.

This work is a result of restructuring an existing Semiconductor Device Simulation package "ALBERTINA", developed at the University of Alberta. The models upon which the restructing was implemented have considered the possible future expansion of the system to encompass a larger variety of devices, and the possibility of other simulation environments.

Acknowledgments

I wish to express my deep gratitude and thanks to my supervisor Prof. Dr. H. P. Baltes, for his constant support, encouragement and insight that made this work possible. I am indepted to my co-supervisor Prof. W. B. Joerg for his support and vigorous discussions that introduced me to the abstract parts of this work.

Many thanks are due to Prof. Dr. R. Lawson for his help and understanding in difficult times.

Special thanks to my friends and colleagues A. Nathan and K. Chau for always having time to assist my decisions regarding this work. I also would like to express my appreciation to D. Briglio, T. Smy, and M. Buchbinder for proof reading this thesis. Thanks are due to Mrs. G. Baltes for her fine touch that gave this work a presentable form.

I finally, wish to thank the Alberta Microelectronic Center for their financial support, and allowing me to use their facilities.

Table of Contents

1. Introduction 1

2. Modeling and Simulating Semiconductor Devices 6

  2.1. The Purpose of Modeling and Simulation 6

  2.2. Modeling Semiconductor Devices 7

  2.3. Simulation Steps 9

  2.4. Previous Works 10

  2.5. Definition and Specifications of User Interfaces 16

  2.6. Albertina, History and Requirements 21

3. Modeling Semiconductor Devices 24

  3.1. Semiconductor Device Basic Equations 24

    3.1.1. Poisson's Equation 25

    3.1.2. Continuity Equations 27

    3.1.3. Carrier Transport Equations 27

    3.1.4. Current Density Equations in a Magnetic Field 30

  3.2. Models of the Physical Parameters 31

  3.3. Doping Profile 32

    3.3.1. Distribution Parameters 32

    3.3.2. Calculation of the Distribution Parameters 34

    3.3.3. Distribution Functions (Doping Profiles) 35

  3.4. Carrier Mobility 38

    3.4.1. Scattering Mechanisms and Theoretical Mobility Models 39

3.4.2. Empirical Mobility Models 41
3.5. Carrier Generation and Recombination 46
3.6. Band Gap Narrowing 50
3.7. Dopant 52
3.8. Boundary Conditions 52
4. The Abstract Models of a User Interface 54
4.1. User Perception of Albertina 54
4.2. Designer Perception of Albertina 58
4.3. Data Structures 61
4.4. Total Object Description 67
4.5. Abstract Versus Applied Perception of Albertina 69
5. Application of the User Interface 71
5.1. Object Model Application 71
5.1.1. Geometry Description 71
5.1.2. Physical Description 77
5.2. Action Model Mechanisms 78
5.2.1. Node Referencing Mechanism (NRM) 78
5.2.2. Tree Browsing Mechanism 80
5.3. System Primitive Operations (SPO) 83
5.4. Application of the Action Model 84
5.4.1. User Primitive Operations (UPO) 85
5.5. Tree Building Mechanisms (TBM) 101
5.5.1. Generic Tree Building Mechanism 102
5.5.2. Iterative Tree Building Mechanism 104

5.6. Summary and Illustration of Concepts 106

6. System Integration 112

6.1. Application of the Albertina's Graphical Layer 113

6.1.1. General Layout 113

6.1.2. Input-Output Considerations 117

6.2. Definition of the Physical Parameters System Calls 121

6.2.1 Invokement of PPSCs 121

6.3. General Application of PPSC 124

6.4. Parameter Extraction 124

6.5. Application of the Physical Parameters System Calls 125

6.5.1. Band Gap Narrowing PPSC (band) 125

6.5.2. Mobility PPSC (mob) 126

6.5.3. Doping Profile PPSC (dop) 129

6.5.4. Generation and Recombination PPSC (genere) 140

6.5.5. The Expansion of OPTIONS 140

6.6. User Interface vs. External Modules 141

6.6.1. Invokement of External Modules 141

6.6.2. Mesh Linkage 142

6.6.3. PDE Solver Linkage 146

6.6.4. The Mesh and PDE Solver Sequence 146

6.6.5 Monitoring and Reliability 148

6.7. Graphics Package 149

6.7.1. Plotting Routine 151

6.7.2. Two Dimensional Geometric Display 151

6.8. Example of a Typical Simulation Session 154

7. Discussion and Outlook 166

References 170

Appendix I : "C" Definition of Nodes 178

Appendix II : Constants of Physical Models 180

Appendix III : Illustrations of Results 183

LIST OF TABLES

Table	Name	Page
3.1	Coefficients of R_p polynomial fit	34
3.2	Coefficients of ΔR_p polynomial fit	35
3.3	Coefficients of the Thurmond Model	52
5.1	Format of "Save"	95
5.2	Sample of saved device	96
5.3	Identification of class 3 node attributes	97
6.1	I/O mechanism truth table	120
6.2	Format of physical parameters data pool	125
6.3	Arguments of mobility PPSC	127
6.4	Format of OPTIONS physical data pool	134
6.5	Sources of doping parameters	135
6.6	Default values for ΔR_l	135
6.7	Arguments of *genere* PPSC	140
6.8	Expansion of OPTIONS	141

LIST OF FIGURES

Figure	Name	Page
4.1	User perception of Albertina	56
4.2	Designer perception of Albertina	60
4.3	An abstract description of a "Region"	63
4.4	Left and right configuration for geometry descriptor nodes	65
4.5	Graphical presentation of Region and Geometry descriptors	66
4.6	Total device description	68
5.1	Circle-Rectangle-Ellipse-Pocket subtrees	74
5.2	Curve-Polygon subtree	75
5.3	Tree presentation of a circle	76
5.4	Device cluster	100
5.5	Albertina's levels of interactions	107
5.6	Albertina's operational model	108
6.1	Application of Albertina's graphical layer	114
6.2	Action-Panel-Menu relationship	116
6.3	Pipe channel to PPSC	123
6.4	Mobility calculation	128
6.5	Plot of Caughey and Thomas doping mobility model	130
6.6	Plot of Caughey and Thomas electric field mobility model	131
6.7	Plot of Scharfetter and Gummel mobility model	132
6.8	Plot of Thornber-Gummel mobility model	133
6.9	Plot of 1_D Gaussian doping profile for B in Si at 100 & 200KeV	138
6.10	Plot of 1_D Gaussian doping profile for B in Si at 150 & 250KeV	139
6.11	Geometry segmentation	144
6.12	Integration of Albertina	147

6.13 Menus of Region Panel 157
6.14 Menus of Device Panel 158
6.15 Listing of a device 159
6.16 Menu of Shape Panel 160
6.17 Listing of device geometry 161
6.18 Ppar- Mobility Menus 162
6.19 Menu of Display Panel 163
6.20 Display of a device 164
6.21 Invoking the Mesh Generator 165
III.1 Projected range of antimony in silicon 184
III.2 Projected range of arsenic in silicon 185
III.3 Projected range of boron in silicon 186
III.4 Projected range of phosphorus in silicon 187
III.5 Standard deviation of antimony in silicon 188
III.6 Standard deviation of boron in silicon 189
III.7 Standard deviation of phosphorus in silicon 190

LIST OF SYMBOLS

Symbol	Name	Unit
$\vec{A}$	Magnetic field potential vector	tesla.cm
$\vec{B}$	Magnetic induction vector	tesla
BTE	Boltzmann Transport Equation	-
C	Longitudinal deformation constant	VAscm^{-3}
$C(x,t)$	Solute concentration	cm^{-3}
$\vec{D}$	Dielectric displacement vector	Coulomb/cm^2
D_n	Electron diffusion coefficient	cm^2/sec
D_p	Hole diffusion coefficient	cm^2/sec
$\vec{E}$	Electric field vector	volt/cm
E_I	Ion implantation energy	KeV
E_{ac}	Conduction band deformation potential constant	eV
E_{av}	Valence band deformation potential constant	eV
E_e	Electron's energy	eV
F	External force	Newton
G	Generation rate	sec^{-1}
$\vec{H}$	Magnetic field	amp/cm
$\vec{J}$	Conduction current density	amp/cm^2
N_D	Donor concentration	cm^{-3}
N_A	Acceptor concentration	cm^{-3}
N_i	Carrier intrinsic concentration	cm^{-3}
NRM	Node Referencing Mechanism	-
$\vec{P}$	Electric dipole moment	Coulomb/cm^2
Q_f	Charge at interface	Coulomb/cm^2
R	Generation and Recombination factor	sec^{-1}
R_p	Projected range	μm
ΔR_p	Vertical standard Deviation	μm

ΔR_l	Lateral standard Deviation	µm
SPO	System Primitive Operations	-
T	Absolute temperature	Kelvin
TBM	Tree browsing mechanism	-
UPO	User primitive operations	-
V_a	Applied voltage	volt
c_n^A	Auger capture coefficient for electrons	cm^6/s
c_p^A	Auger capture coefficient for holes	cm^6/s
f	Number of electrons and holes per unit volume	cm^{-3}
f_o	Quasi thermal equilibrium distribution function	-
h	Plank's constant	eV.sec
k_B	Boltzmann's constant	$1.38 \times 10^{-23} \frac{Joule}{degreeK}$
k	Wave vector	cm^{-1}
m^*	Carrier's effective mass	kg
n	Electron concentration	$electron.cm^{-3}$
n_i	Intrinsic carrier concentration	cm^{-3}
n_{ie}	Effective intrinsic carrier concentration	cm^{-3}
p	Hole concentration	cm^{-3}
q	Electron charge	1.62×10^{-19}Coulomb
r_H	Hall ratio	-
u_i	i-th central moment	-
u_l	Velocity of a longitudinal acoustic phonon	cm/sec
v^{sat}	Drift saturation velocity	cm/sec
α_n^i	Electron ionization coefficients	cm–1
α_p^i -	Hole ionization coefficients	cm^{-1}
α_y	Yamaguchi's fitting parameter	-
α_{op}	Polar constant	-

α_t	Thurmond's first fitting parameter	-
β	Kurtosis	-
β_p	Kurtosis of the Pearson function	-
β_t	Thurmond's second fitting parameter	-
ΔE_G	Band Gap Narrowing	eV
ε	Permittivity tensor	farad/cm
γ	Skewness	-
Γ_T	Thornber scattering rate scaling factor	-
γ_T	Thornber momentum scaling factor	-
μ_x^y	Drift mobility for particle x, model y	cm^2/volt-sec
μ^*	Carrier Hall mobility	$tesla^{-1}$
ν_e	Phonon's frequency	sec^{-1}
ϕ	Electrostatic potential	volts
ϕ_b	Built-in potential	volt
ϕ_n	Electron quasi-Fermi potential	volt
ϕ_p	Hole quasi-Fermi potential	volt
ρ	Electric charge density	Coulomb/cm^3
τ	Relaxation time	sec
τ^o	Carrier lifetime	sec
ω	Phonon angular frequency	sec^{-1}

1. Introduction

The semiconductor integrated circuits industry has witnessed a rapid growth over the last few decades. As a result smaller and more reliable integrated circuits are realized. The miniaturization of the integrated circuits has gone from Large Scale Integration (LSI) to Very Large Scale Integration (VLSI). The current state of the art deals with circuits of sub-micron dimensions named Ultra Large Scale Integration (ULSI). These technological leaps have resulted in an enormous interest in modeling and simulating, the main facets of the semiconductor industry. Modeling is mainly concerned with developing an expression to describe the behavior a particular system while simulation is the activity of numerically solving the model. Hence, several software simulation packages have been developed. These packages could be divided into three main groups, namely:

1. Process simulators : Such packages simulate the steps leading to the creation of an integrated circuit or a single semiconductor device (e.g. etching, oxidation). Generally, the final results are the device's geometric profile and the distribution of the implanted species within the device.
2. Device simulators : In such packages the behavior of the charge carriers within a semiconductor device are simulated. The behavior is simulated under various conditions (e.g. applied voltages, magnetic field). The final results are the device's current density and the electrostatic potential distribution within the device.
3. Circuit simulators : Such packages are concerned with the simulation of a combination of several semiconductor devices (e.g. logic simulators). In general the final results are AC and DC levels of voltages and currents.

For each of the above groups, many successful simulation packages have been separately developed. Collectively, researchers have mainly concentrated on the validity of the simulation models and the numerical techniques used in solving the models. Yet, an important aspect of using such packages has been neglected, namely, *package* *interface*. A simulation package is required to interface with two environments. The first is the human user. The second is to another simulation package i.e. interfacing package A to package B, where each belongs to one of the above groups. Although computer-user interface has been studied for quite a period of time, no specific implementation has been done to accommodate semiconductor simulators. This work attempts to fill such a gap by presenting a novel interface to a semiconductor device simulation package. This interface is designed and implemented according to the following specifications :

1. To interface users with the package as a whole. According to each user's level of expertise the interface should offer the following main features :
   a. The convenience of accessing any of the package's simulation steps.
   b. The freedom of invoking simulation steps, according to a sequence set by the user.
   c. The ability to choose the appropriate parameters and models describing the simulated device.
   d. The ability to inspect and display results.
   e. The ability to modify input data or model parameters, if necessary.
2. Interfacing the different modules forming the simulation package.

The structure of the interface is designed so as to easily accommodate future expansions. Such expansions could be, interfacing other simulation packages, or the enhancement in terms of new numerical techniques and more accurate models.

The implementation of this thesis is applied to an existing device simulation package "ALBERTINA". As a result groups of simulation steps where gathered into isolated modules. The modules directly correspond to the steps of a typical simulation session, namely, description, discretization, solution and display of the simulated device and results. In addition to Albertina's lack of an interface, the description and display modules were rigid and relied on external packages. Therefore, two new modules to accomplish the tasks of description and display where developed in order to enhance the performance of Albertina. In addition to the user interface, these two modules are incorporated in this work. The description module is composed of two parts. The first part allows the user to describe the geometry of the simulated device. The second is a physical module which allows the user to define certain parameters pertaining to the models of the semiconductor device. The parameters are conventionally known as the *physical parameters*. The physical module contains a library of models for each parameter. The user can select and allocate models to regions of the simulated device. The description of the simulated device is dynamically stored in versatile tree structures. The construction and manipulation of such a tree is implemented by a set of well defined operations. The third module developed for Albertina is basically a graphics module. The module is used to plot and display the geometric and physical data.

This work has been implemented on a SUN 3/160 workstation in a UNIX† programming environment. The programming language "C" was used. The user interface used Sunview tools. The graphics package incorporated GKS, SUNCGI and SUNCORE.

The thesis is organized as follows :

Chapter 2 provides background material. We start by defining modeling and simulation and how they are related. We then proceed to describe the models that formulate the behavior of semiconductor devices. The general steps of simulating a semiconductor device follows. In the course of this work a literature survey of device simulators was conducted. A summary of this survey is provided in section (2.4) of this chapter. Chapter 2 ends with a general discussion on user interfaces, their definition and specifications. This discussion is a compilation of another literature survey conducted to investigate user interface state of the art.

---

† UNIX is a trademark of Bell Laboratories.

Chapter 3 is a detailed discussion of the semiconductor device models. It starts with an elaborate derivation of the basic semiconductor device equations (models). These equations describe the behavior of the device as a whole. Next, we present a detailed discussion of the physical parameters. The discussion includes the different physical mechanisms governing the parameters and their various theoretical and empirical models.

In chapter 4 we present the abstract ideas upon which this work is based. We start with a quick summary of user interfaces. We then proceed to discuss the abstract ideas of Albertina's user interface. These ideas are divided in two parts. The first deals with the user's perception of the simulation package, the second discusses the designer's point of view. We end the chapter by correlating both views.

Chapter 5 presents in detail the implementation of the user interface in terms of the abstract ideas presented in Chapter 4. This implementation allows the user to describe the geometry and physics of the simulated device, thus accomplishing the description modules. The modules and their associated mechanisms are presented in detail. The algorithms used to construct the modules are presented in a pseudo-Pascal form. Chapter 5 ends with a summary and illustration of the main concepts.

Chapter 6 presents the integration of the simulation package. First, we present the implementation details of the user interface on the Sun workstation. The user interaction techniques with the description modules are discussed, followed by the methods used to invoke other simulation steps. We then proceed to describe the mechanisms we implemented to calculate numerical values for the physical parameters. The implementation of the graphics package developed for Albertina is then described. Chapter 6 ends with an example of a typical simulation session, where a Hall device is described, displayed and discretized.

Chapter 7 is a summary of the thesis and a discussion of future work and possible improvements.

Appendix I gives a list of the data structures used to implement the description modules. The list is provided in the programming language "C". In appendix II, the values of various constants pertaining to the physical parameters are given. Plots of some physical parameters are provided in appendix III.

2. Modeling and Simulating Semiconductor Devices

In this chapter we first discuss the basic definitions of modeling and simulation and their relationship. We then proceed to describe the equations that model the behavior of semiconductor devices. The steps of simulating a semiconductor device are then summarized. Next, we present a summary of the literature survey conducted to assess the state of the art of semiconductor device simulators, followed by a similar discussion with respect to user interfaces. Finally, a brief analysis of Albertina is given.

2.1. The Purpose of Modeling and Simulation :

The term "Modeling and Simulation" abstractly describes the complex activity of constructing models of a natural system and simulating them on a computer. Modeling deals with creating a valid mathematical representation of the real system. The art of modeling could be summarized in three main steps namely, acquisition, simplification and validation. In general, models are acquired either through theoretical derivations or empirical techniques, and are often presented in terms of differential equations. Next, simplifying assumptions are applied in order to reduce the model's complexity. Finally, a model is considered valid when it meets the following three criteria [1] :

1. The model output matches existing data from the real system.
2. The model can forecast results of possible measurements.
3. The model reflects the behavior of the real system, in terms of providing data that are difficult to measure.

The process of simulation deals with the techniques of numerically solving the simplified model. The first step of this process is to define the model's application domain. Then through iterative, recursive or inductive steps the model is applied to a given set of data in order to produce a consistent solution. Simulations are carried out in order to :

1. Decrease the number of trials and errors of producing the simulated system.
2. Reduce production cost.
3. Provide further insight into the simulated system.

The benefits of simulation can be exemplified in the simulation of ion implantation [2]. In this case simulation is used to plot the trajectory of beams of ions incident on the crystal lattice of the target material. The trajectory is a function of the orientation angle and the energy of the ion. In the course of simulation study it was noted that under certain circumstances ions penetrated to a much greater depth than expected. Close study revealed the existence of an unknown phenomenon, *channelling*, which was subsequently confirmed experimentally, thus further insight to the simulated system was provided.

Another significant benefit is the cost of simulation versus the cost of prototyping the simulated system. The cost of simulation is constantly decreasing due to the decrease in price of computers and memory. Finally, the increase in computer power and the further sophistication of numerical and computational techniques allows even better, faster and more cost-effective results.

2.2. Modeling Semiconductor Devices :

The origin of semiconductor device modeling using numerical instead of analytical methods can be traced back to the early works of Gummel [3] on a one-dimensional bipolar transistor. Gummel's algorithm has since been extensively used in its original form, and it has been extended in two dimensions by others [4, 5].

Modeling the behavior of semiconductor devices is concerned with formulating the motion of charge carriers under the influence of local and external electromagnetic fields. The charge carriers are the electrons and holes. Semiconductor devices are modeled by a set of partial differential equations. These models are derived from

Maxwell's equations and Boltzmann transport equation. The models are Poisson's equation, continuity equations and carrier transport equations. In this section we provide a brief description of these equations, as follows :

1. Poisson's equation : This equation relates the distribution of the electrostatic potential ϕ and the space charge density ρ. Poisson's equation is derived from Maxwell's second equation which describes the relation of the electric field vector $\vec{E}$ and the magnetic field induction vector $\vec{B}$. The scalar term ϕ is introduced in order to decouple the magnetic and electric fields. The space charge density ρ represents the algebraic sum of the electron, hole, donor and acceptor concentrations.

2. Current continuity equations : The current flowing in a semiconductor device is due to the motion of electrons in the conduction band and holes in the valence band. The current continuity equations describe the conservation of the charge carriers within the device. Hence, this groups contains two equations, one per carrier. Each equation relates the carrier current density J, the carrier concentration rate of change with respect to time and the rate at which the carrier is generated or recombined.

3. Carrier transport equation : This set of equations is an approximated first order solution of Boltzmann's Transport Equation (BTE). The BTE expresses the effect of electromagnetic fields on the distribution of charge carriers. The solution of BTE results in two equations, one per carrier. Each equation attributes the current density to two physical mechanisms, namely, drift and diffusion. In the presence of an electric field $\vec{E}$, the charge carriers are subject to acceleration. The motion of the carriers due to this mechanism creates the drift component of the current. Drift current density is a function of the carrier mobility parameter μ, and the carrier concentration. Diffusion of carriers between two points occurs when the carrier concentration gradient between them is not equal to zero, thus, creating the

diffusion component of the current. This component is a function of the diffusion coefficient D, which in turn is a function of the mobility μ. The presence of the device in a magnetic field will subject the carriers to a deflective force known as the Lorentz force. Therefore, a third component is included in the transport equations to count for this effect. This component is a function of the Hall mobility μ^* and the vector product of the current density and the magnetic field induction.

The models discussed above are functions of certain parameters (e.g. mobility μ), known as the *physical parameters*. In turn these parameters are functions of various physical quantities and mechanisms. Hence, theoretical and empirical models have been developed for them. A detailed discussion of the models is given in chapter 3.

2.3. Simulation Steps :

In the previous section we have described the basic equations of semiconductor devices. In this section we briefly outline the semiconductor device simulation steps. As per section (2.1) simulation means numerically solving the above models. The analytical approach to such a solution is summarized in the following steps [6, 7, 8, 9, 10] :

1. Device definition : A simulation process starts by defining the model's domain. In our case the domain is the geometric and physical description of the simulated device. In principle semiconductor devices are three dimensional structures. The symmetry of the problem allows its reduction to two dimensions. Thus, describing the device involves describing the geometry in a two dimensional Cartesian plane. The device as a whole is envisioned as a group of adjacent regions, where each region has it's own geometric and physical properties. Accordingly, a simulation session usually starts by the user entering the geometric and physical description of the simulated device. The description is entered region by region. This preliminary step is provided in this work.

2. Device discretization : The partial differential equations modeling the semiconductor device derived in the previous section cannot be solved in closed form. Therefore, the numerical approach is to approximate the equations in each device domain by a set of non-linear algebraic equations. Next, the device domain is discretized. The discretization involves partitioning the geometric domains of the device into a finite number of subdomains. The result is a large system of non-linear algebraic equations. The unknowns of the equations are approximations of the continuous variables at discrete points. The discretized device is graphically represented by a mesh composed of a large number of linked nodes. The techniques used in this step are Finite Box, Finite Differences or Finite Elements. Further discussion of this step is found in [11, 12].
3. Solution of non-linear algebraic equations : This step involves solving the algebraic equations for each mesh point. The solution is started with a good estimate of the initial values of the unknown variables. Next, iterative techniques (e.g. Newton's method) are applied to the variables. Convergence of the iterative techniques can often be accelerated using certain methods, e.g. Successive Overrelaxation Method (SOR). The solution of this system can then be used to evaluate the device characteristics.

2.4. Previous Works :

In this section we present a summary of the literature survey conducted to assess the current state of semiconductor device simulators. In the discussion we mainly concentrate on two aspects, the user interface and the physical parameter models available within each system. We note though, that in the available literature only few papers have described either aspect. In most publications the structure of the simulation package and the user interaction techniques are vague and incomplete. One of the recent publications [13], makes it evident that there is a great need for versatile interfaces. The task of such interfaces is to mainly provide a man-machine interface. It is also required to interface the simulator with other simulation packages.

In general, semiconductor simulation packages are described in terms of scattered black boxes that are completely governed by the user, who is then forced to keep track of the input and output of each of these boxes. The geometric and physical description of the device are directly stored in large data files, thus, reducing the system's flexibility and increasing CPU time consumption.

In [14], a package capable of solving a wide variety of semiconductor device problems is described as a hierarchy of three levels: On the bottom are a number of mathematical software libraries (e.g. sparse matrix software). The middle layer is a large assemblage of general purpose finite elements methods and graphics programs. The top layer, with which the user interacts may be one of several layers, such as a general purpose interactive layer used for "one shot" problems. The user has to provide a few subroutines to define the PDE solver and the boundary conditions. The geometry is entered in terms of a union of triangular shapes. The package is written in portable FORTRAN, where machine dependent parameters are isolated. Graphical output must be specified before the computer is started, which slows the process drastically. Each type of problem solved requires that a specific solution strategy be developed.

HFIELDS [15, 16] developed at the University of Bologna, is another simulation package that we have studied. We note that this is one of the few packages where the system structure and data flow are discussed at some length. The package is written in FORTRAN and is divided into three blocks:

1. Input section and the grid generator: This section consists of six modules named G, V, P, C, S, R and a set of general purpose routines called O. Each module is responsible for the description of some aspect of the device. The modules are then applied to the description in a certain sequence. The output of each module is stored in a file which is used as an input by the next module. The files are named as *Xyyyyy,* where X is the module's name, and yyyyy is a user defined five character string. Module G defines the geometry of each zone as a piecewise linear curve, where the description has to be closed i.e ($x_1 = x_n \;\&\; y_1 = y_n$).

Module V retrieves data from Gyyyyy and assigns Dirichlet boundary conditions to selected boundaries. The result is stored in Vyyyyy. Module P uses the data in Gyyyyy to define the physical parameters to each zone and stores them in Pyyyyy. Module C defines some parameters used by the grid generator and stores them in Cyyyyy. Module S defines parameters and commands for the solver (e.g. convergence check) and stores them in Syyyyy. Module R is the grid generator; it uses the data in Gyyyyy, Pyyyyy and Cyyyyy to create a triangular grid. Module O comprises general purpose routines which connect the modules and can be called by any of the modules. The tasks of the routines are :

a. Internal file management.

b. Setting up of the simulated device's common areas.

c. Definition and display of menu titles and messages.

d. Syntactical input control.

The grid generator module creates a triangular mesh based on the data from Gyyyyy, Pyyyyy and Cyyyyy.

2. The solver: This second block of HFIELDS is responsible for solving the non-linear algebraic equations.

3. The output section: Through a menu organization with an additional help capability, this section retrieves the output data of the solver and performs the following tasks:

   a. Drawing a 3D plot of a variable.

   b. Drawing a contour plot.

   c. Drawing a 2D plot of a variable.

   d. Drawing a 2D set of arrows representing the distribution of a vector field.

The third simulation package we reviewed is briefly discussed in [17]. This software environment uses modularity and data structure management. This is done by using a "PASCAL - like" data structure that was made compatible with FORTRAN by writing a library of data management routines. Through this library geometry information could be accessed by name. Data structures may be declared, instantiated, transferred, deleted, and copied into files. In a later publication [18], the authors provided more information with regard to :

a. Data structures: Using FORTRAN has forced the developers to create DAMSEL, a data management system. DAMSEL is used to manage what the authors defined as "*skeletons and boxes*". Each skeleton and box is an analogy to a "C" structure or PASCAL record definition.

b. User Interface : The user's interaction with the system is through an "Input command language" allowing parameter lists to be entered. The input language has two forms of similar syntax. These forms are used in the following manner :

   i. Data cards: This form allows the user to enter data to the simulator and is interpreted by a general purpose command analyzer.

   ii. Commands : Through this form the user can call some specific routines (e.g. the mesh generator).

c. Geometric manipulations : The third part of this system provides certain terms that are used to describe the device geometry. These terms are *bounds,* describing the region's boundary conditions; *faces,* a list of four bounds compose this term which is used to describe a two-dimensional device region. *Volumes,* this term is used to describe three dimensional regions, the description is defined as a list of faces.

An integrated system MECCA is described by Prendergast [19]. This system consists of device and circuit simulators. The output is a set of circuit-level model parameters (i.e. currents and voltages). The input is a set of device geometries which is used by the process simulator to generate impurity doping profiles. The device simulator uses the profiles to generate a set of characteristics needed for parameter circuit extraction. A measurement system is then used to generate a similar set of characteristics. These measurements are used to verify the output of the device simulator. From the verified output AC and DC circuit parameters are extracted. MECCA uses MEDUSA [20] as a circuit simulator, where a variety of process simulators were incorporated to provide models for ion implantation and other device processing steps. The development of MECCA was basically concerned with parameter extraction and module interfacing. In [19], the author has not discussed in any detail the method through which the user-system interface has been implemented. We mention it as a good example of a growing trend in the field of semiconductor device simulation, namely interfacing process-device-circuit simulators.

Another example of interfacing process and device simulators is described by Hamilton and Amaratunge [21]. In this system, MINIMOS is used as a device simulator after adapting the source code to allow its interface to a process simulator. The interface is a simple subroutine where doping profile models are defined to regions where interfacing is inapplicable.

Lombardi et al. [22] introduced IDAS as a process oriented software environment. The user has easy access to all necessary tools for analyzing and developing advanced semiconductor structures. The tools are described as:

a. Pre-processor or input modules.

b. The processor or solver.

c. Post processor or output module.

The input and output modules are considered as the user interface to the solver. The user is offered a high-level command language which is translated to the solver data structure. To enhance the user interface a menu-driven approach is used. The menus display operations, program status, available commands, and parameters with their default values for the users selection. Furthermore the user has access to a command history, on-line-help, and the ability to create macros. In IDAS the complexity of the input and output modules is closely connected to the solver capabilities. The more general the solver, the more complex are the user interfaces, especially the input part. Device geometry, impurity concentration and geometry discretization are organized by the input module. Device geometry is defined through a consecutive application of process-like steps on an initial semiconductor substrate. The result is a complicated description of the problem to be solved. A simpler geometric description is achieved through entering straight lines. In this environment impurity profiles are obtained from process simulators and handled as analytical functions. The generation of the initial mesh uses previous solutions of impurity and potential distribution. The mesh is refined upon user command. The solver is then run and is successful only if a large number of cross-checks, warning and error messages have been issued in the above phases. Output is an application of *GKS (Graphics Kernel System)* which is mandatory to analyze complex variables such as potential distribution and current densities. The display is in terms of 1D, contours, arrows and 3D plots.

We finally wish to give a quick review of MINIMOS, a popular two dimensional MOS transistor analyzer developed by Selberherr et al. In [23] MINIMOS was described with the following features:

1. Band gap narrowing has not been accounted for.
2. The intrinsic carrier concentration is constant.
3. Permittivity is homogeneous.
4. Recombination and generation are ignored.
5. Only channel carriers contribute to current flow.
6. The temperature distribution is homogeneous and can be varied within a range of 250-450K.
7. The doping profile is obtained from either one of three sources :

a. MINIMOS itself, where the dopant distribution are analytical expressions in closed form.

b. The process simulator SUPREM.

c. Defining the doping profile point by point.

In MINIMOS, Poisson's equation is linearized as an elliptic differential "Helmholtz" equation. This equation is solved using a nonuniform automatically generated mesh via finite differences techniques. The mesh has a minimum size of 25 X 25 nodes to a maximum of 60 X 60. The potential drop between any two neighboring nodes is maintained at less than 10 thermo-voltages ($\frac{k_B T}{q}$). The carrier mobility model originally offered in MINIMOS has accounted for lattice, impurity and surface scattering, and for the influence of velocity saturation. The models were combined using the Mathiessen rule (see section 3.4) with a weight 2. The mobility models were later enhanced [24] to offer more accurate models and to account for hot carrier transport.

2.5. Definition and Specifications of User Interfaces :

Over the last few decades application and use of computers has grown phenomenally. A parallel growth of complexity and sophistication of such machines is a natural result. Due to these facts, there is a need to make such computer systems easy to use, i.e. *user friendly*. This need if often accommodated by a software package known as the *user interface*. The main function of such interfaces is to translate the user's perception of the system to its actual structure within the machine. The main problem facing the design and implementation of such packages is the high complexity of user-computer interactions. Hence, modeling such interactions is very difficult. Most interaction models are based on practical experience in a specific context rather than being based on well established theories. In this section we present a discussion of proposed user interface definitions and specifications. The following is a result of our attempt to investigate user-interface state-of-the-art, it is focused on certain specifications that are relevant to this work. Literature that is more of an abstract nature is not included.

In [25], the author states that the design of a user interface has to be with respect to the intended user. Thus, three categories of computer users are identified. Each category of users and their requirements are discussed as follows :

1. Specialist users : The tasks performed by such users are concerned with finding an elegant way to access and manipulate the internal structure of the computer system. User interface designers fall under this category. The main requirements of this group are a general purpose interface, an appropriate programming language and a convenient debugger.
2. Professional users : The applications of such users are of a relatively complex nature (e.g. simulation and data analysis). Hence, the user will frequently use an input device either to enter or to modify data. For this reason, selecting appropriate and comfortable input devices and mechanisms are important. A well designed interface will allow users to enter their data with minimum control and maximum ease while using the full terminal screen.
3. End users : Generally, such users perform simple tasks that require one decision at a time (e.g. clerical staff in an office). The interface should provide communication of plain language syntax. Most or all of the processing details are usually hidden. Graphical presentation of operations provided by the system are the current trend.

In this paper the requirements of a user interface are generalized as ; "being able to create an intelligent dialogue between the user and the system". Where the dialogue is performed in order to teach, train, advise or interview a user. Where interviewing is collecting information from the user.

Enderle [26] describes models, methods, principles, and tasks for the design of user interfaces. The description is done within the context of creating a user interface for a Computer Aided Design (CAD) application. CAD relies on computer graphics for input and output purposes and is a favourable area for user interface design. Thus the term an interface of a *graphical nature* is used. First, three main problems of user interface application are stated as follows :

1. The configuration of the interface with respect to a specific application.
2. The responsibility for creating and managing graphical output.
3. The services and functions available at the communication level between the interface and the application.

The author proceeds to describe three different interface structures. We mention only one, which is of direct interest to this work. In this structure the application, interface and graphics packages are on one level. They are parallel processes that are synchronized by multitasking facilities. In this paper, the interface configuration components with respect to a specific application are defined. These application oriented components are used to create a "dialog". The dialog is defined as ; The actual interactive process between the user and the system. The components of a user interface are listed as :

a. Menus and screen layout.
b. Error messages displayed to the user.
c. Prompts, prompting user for optional or mandatory data.
d. Icons, where an icon is a graphical symbol used by the user for communication.
e. List of task names.
f. List of variable names.
g. Mechanisms for interface-application communication.
h. Selection of interaction techniques, where a number of techniques are offered.

In another publication [27], features of smart user interfaces are given. Such interfaces are required due to the fact that "users are more likely to make an error when they try to re-execute a failed command than when they tried to execute it" [28]. A smart user interface is defined as an intelligent intermediary between user and system where user modeling helps to provide a personalized answer. The main features of such a user interface are as follows:

1. Modularity, to facilitate future modification.
2. Applicability to many systems and system utilities.
3. Finds user's misconceptions and corrects them by giving suitable instructions.
4. Transparent except when needed.
5. Provides feedback, so that users know what is happening.
6. Novice users are allowed to choose commands by recognition rather than by recall - a menu is provided.

The author implemented such an interface to the Unix operating system. Unix was chosen because of its enigmatic response and the availability of a large group of users. Furthermore, Unix is most likely to be used by a wide range of users in the future, in commercial as well as research environments.

Another user interface for a Unix operating system is described in [29]. A graphical system is supported by multiple and independent areas for graphics interaction on a single screen. Such areas are known as *windows*. This required the development of a *window manager system*. The design of such manager systems involves the user interface, the internal system software and the operating system. The user interface decides how the user will arrange the window on the screen, and it ensures that application programs make use of dynamic graphics techniques. System software determines the structure of the application programs, their display in multiple windows, and multitasking. The authors state that the window manager, the user interface and system software are eventually merged in order to overcome certain constraints (e.g. multitasking). The interface was implemented on a Whitechapel MG-1 graphics workstation.

Reichman [30], agrees with the idea of user interface and application merger. In this paper the author discusses the evolution of computer-user interaction. The evolution was basically a result of shifting from batch-processing to interactive programming. User-computer interaction was divided in three forms as follows :

1. Indirect interaction : In the early days of computers a user interacted with the computer through a paper teletype. Separate instructions were needed to edit and to see the effect of that edit. The time delay between the user input and the computer output was large. The user could not *see* what the computer was doing, and the existence of a barrier was felt. In this form of interaction the user interface and the computer were separate.
2. Limited direct interaction : This form of interaction was due to a large hardware development, which supported full screen display on a CRT terminal. Thus, it became possible to show results immediately and automatically. Software development was slow in following these changes. Moreover, it took time for users to change their interaction habits. This communication form between the user and the computer was more direct, but was still limited. Users could interact with only one process through *single-stream* systems. This from of interaction breaks the natural flow of thoughts.
3. Direct interaction : This is the most recent form of user-computer interaction. The current technology provides support systems for windows - dividing one screen into multiple virtual screens. Within each window the user can invoke a different process, thus making it easy to move among different processes. In this interaction form the user simply moves the cursor from one window to another using a mouse. In multiple window systems the computer and the interface are no longer separate processes. They are composed of a set of interactive processes, many of which are directly visible to the user. This form of interaction is more natural and supportive of the way humans work.

Finally, in [31] the user interface is defined as a medium through which a user could display the complex information structure of a computer system. Such a definition required the specification of where and how to display this information. The authors suggest that the display would be in a system of multiple windows. The user is provided with a mechanism to select different parts of the system to be displayed. The publication hints to a futuristic view of user interfaces based on increasing the tasks delegated to the computer. The role of the user in such a case would be defining the goals and the system would perform the tasks. Thus, interpretation, strategy, inference, implementation and other aspects of solving a problem would be the responsibility of the interface and the system.

As a result of the above survey it is evident that user interface state-of-art features are mainly implementing it of a graphical nature, and an increasing tendency to increase its responsibilities and intelligence. Furthermore, the implementation is carried out with intention to serve a certain class of users in a certain application domain.

2.6. Albertina History and Requirements :

Semiconductor device simulation is currently an important and growing area of research. Comprehensive packages are being developed at several universities and industry sites.

Albertina is a device simulation project undertaken at the University of Alberta. It was initiated in 1983, with focus on semiconductor sensors, magnetic in particular. In this section we discuss the problems that faced Albertina.

The current user session of "Albertina" is an *ad hoc* process that requires a certain degree of expertise to use and has severe limitations (e.g. only rectangular or union of rectangular shapes are handled). The system is scattered amongst several computers. Furthermore different simulation steps have to be explicitly invoked by the user in a particular sequence (e.g. mesh generator, PDE solver .. etc). The above steps are tedious, error prone and may risk inconsistent solutions.

In order to enhance the performance of Albertina achieve the goals of generality and expansibility, it requires the following :

1. A powerful user interface, with the following capabilities :
    a. Based on well defined and simple data structure provide the user with a facility for defining the device geometry. The data structures should allow the description of a large variety of shapes, namely, rectangular, elliptic, circular, polygonal and curved. Yet, the structures should maintain a certain generality to permit future expansion.
    b. To implicitly invoke different solution modules. The interface will assume the responsibility of providing the necessary parameters and checking information gaps. Such gaps would then be filled either by the user or the system itself. Such modules could be on different and remote machines due to "Albertina"'s distribution, the interface would then provide the appropriate communication protocols.
2. A library of physical models and parameters : The library is composed of valid models for each of the physical parameters pertaining to the semiconductor device basic equations. It should be accompanied by tools that would allow the user to conveniently designate any of the models to a device region. A versatile mechanism to provide numerical values for each of these parameters in accordance with the model is also required.

In this work we have restructured Albertina in order to provide an experienced user with these features. An experienced user requires the ability to interactively define the device and guide the solution process by selecting areas, parameters or modules and applying either a value or a procedure to the selected item. In other words, an experienced user requires a set of operations at a certain level of functionality to provide a high degree of freedom to manipulate the simulated device. A detailed discussion of the ideas and techniques used to restructure Albertina is provided in the following chapters. The sequence of this discussion is; Chapter 3 provides a further detailed discussion of the physics of semiconductor devices, where the basic models are derived and the physical parameters are identified and modeled. Chapter 4 provides our abstract approach to restructuring Albertina. In chapter 5 the implementation of this approach is formulated and demonstrated. The integration of the various aspects of this work and further implementation details are provided in chapter 6. The text ends with chapter 7 where conclusions and outlook are provided.

3. Modeling Semiconductor Devices

The equations describing the behavior of semiconductor devices were briefly described in section (2.2). In this chapter we present a more detailed mathematical and physical description of these equations. We start with the methods and assumptions used to derive the main models of semiconductor devices. The resulting equations are known as, Semiconductor Devices Basic Equations (SDBE). In section (2.2) we have mentioned the SDBE parameters known as, physical parameters. This chapter includes the details of these parameters, their mechanisms and models. In the implementation of Albertina these models are presented as the "Library of Physical Parameters".

3.1. Semiconductor Device Basic Equations :

Modeling and simulating a semiconductor device is an attempt to characterize an arbitrary semiconductor structure under various operating conditions. In this process we must keep in mind that the electronic properties of semiconductor devices are related to the motion of charge carriers under the influence of external and local fields. The models describing the above are known as "Semiconductor Device Basic Equations" and are derived under certain assumptions from Maxwell's equations [6, 7] namely :

$$\nabla \times \vec{H} = \vec{J} + \frac{\partial \vec{D}}{\partial t}, \tag{3.1}$$

$$\nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t}, \tag{3.2}$$

$$\nabla . \vec{D} = \rho , \tag{3.3}$$

$$\nabla . \vec{B} = 0, \tag{3.4}$$

where:

$\vec{B}$ = Magnetic induction vector.

$\vec{D}$ = Dielectric displacement vector.

$\vec{E}$ = Electric field vector.

$\vec{H}$ = Magnetic field vector.

$\vec{J}$ = Electric current density vector.

ρ = Electric charge density.

In the following sub-sections we discuss the detailed derivation of the semiconductor basic equations, namely, Poissons's equation, the current continuity equations and carrier transport equations. The discussion includes the assumptions under which the equations are derived.

3.1.1 Poisson's Equation :

A relation between the electric field vector $\vec{E}$ and the electrostatic potential ϕ is obtained through certain manipulations of Maxwell's third equation (3.3). Its application to semiconductors is done achieved introducing the following relation between the electric field $\vec{E}$ and the displacement vector $\vec{D}$:

$$\vec{D} = \varepsilon\vec{E} + \vec{P}, \tag{3.5}$$

where

ε = Permittivity tensor.

$\vec{P}$ = Electric dipole moment per unit volume.

Equation (3.5) is valid for all materials where the permittivity tensor is time invariant. The validity is subject to the dipole moment being negligible when operating in the limit of relatively small electric fields and frequencies not close to the dipole vibration frequency.

Next, a vector field $\vec{A}$ is introduced in the following relations :

$$B = \nabla\times\vec{A}, \tag{3.6}$$

$$\nabla . \vec{A} = 0 . \tag{3.7}$$

Therefore, by substituting equation (3.6) in Maxwell's second equation we get :

$$\nabla \times (\vec{E} + \frac{\partial \vec{A}}{\partial t}) = 0 , \tag{3.8}$$

which can further be expressed as a gradient field by introducing the scalar quantity ϕ as follows :

$$\vec{E} = -\frac{\partial \vec{A}}{\partial t} - \nabla\phi . \tag{3.9}$$

In quasi-static conditions equation (3.9) is reduced to :

$$\vec{E} = -\nabla\phi . \tag{3.10}$$

Therefore substituting equation (3.10) in (3.5) we get :

$$\vec{D} = -\varepsilon\frac{\partial \vec{A}}{\partial t} - \varepsilon\nabla\phi . \tag{3.11}$$

Substituting the expression (3.11) of $\vec{D}$ in equation (3.3) we get :

$$\nabla(\varepsilon\frac{\partial \vec{A}}{\partial t}) + \nabla(\varepsilon\nabla\phi) = \rho . \tag{3.12}$$

From the above definitions of $\vec{A}$ and ϕ, the first term of equation (3.12) is reduced to zero. Hence, assuming that the permittivity ε is homogeneous, we reach the Poisson form :

$$\nabla(\varepsilon\nabla\phi) = -\rho . \tag{3.13}$$

For a semiconductor the space charge density ρ is further formulated as :

$$\rho = q(p - n + N_D - N_A) \tag{3.14}$$

where n and p are the electron and hole carrier densities, and N_A and N_D are the electrically active donor and acceptor concentrations.

We therefore reach the final form of Poisson's equation that is used for semiconductors:

$$\nabla^2\phi = \frac{q}{\varepsilon}(n - p + N_D - N_A)\,. \qquad (3.15)$$

3.1.2. Continuity Equations :

In semiconductors current density is the sum of J_n and J_p expressing the motion of electrons in the conduction band and holes in valence band respectively. Therefore,

$$J = \vec{J}_n + \vec{J}_p\,. \qquad (3.16)$$

The continuity equations express the charge conservations within the device and are derived by ~~taking the~~ divergence of Maxwell's first equation (3.1) and incorporating Maxwell's second equation, i.e.

$$\nabla.\nabla \times H = \nabla\vec{J} + \frac{\partial\rho}{\partial t} = 0\,. \qquad (3.17)$$

Assuming that only the mobile carriers are time variant, we rewrite equation (3.17) as :

$$\nabla.\,(\vec{J}_n + \vec{J}_p) + q\frac{\partial(p-n)}{\partial t} = 0\,. \qquad (3.18)$$

By introducing the function R describing the net generation (-R) and recombination (+R) of electrons and holes, and treating n and p as independent variables (3.18) is decomposed into:

$$\nabla\vec{J}_n - q\,\frac{\partial n}{\partial t} = qR\,, \qquad (3.19)$$

$$\nabla\vec{J}_p + q\,\frac{\partial p}{\partial t} = -qR\,. \qquad (3.20)$$

The above equations are known as the current continuity equations.

3.1.3. Carrier Transport Equations :

In thermal equilibrium, the distribution function giving the probability of occupation of various energy levels is determined by the Fermi-Dirac distribution function [6, 7]. When electromagnetic fields are applied or in the presence of a thermal gradient carriers are redistributed. In this case the distribution function is expressed by Boltzmann's Transport Equation (BTE) [7]; this equation essentially describes the electron motion as a result of a sequence of drifts in the electric field followed by scattering events due to particle collision i.e.

$$\frac{df}{dt} = \frac{\partial f}{\partial t} + u.\nabla f \pm \frac{q}{h} E.\nabla f = (\frac{\partial f}{\partial t})_{coll} \,, \tag{3.21}$$

where f is the number of electrons and holes per unit volume at a time t and u is the group velocity.

BTE represents a continuity equation in the phase space. It is valid for a single particle under the following approximations :

1. Interaction among carriers is weak.
2. The scattering probability is independent of external fields.
3. Collision duration is negligible with respect to the particle's flight time.
4. The band theory and effective mass theorem apply to the semiconductor.
5. Collisions are completely elastic, i.e. particles do not suffer any energy loss.
6. The electric field variation is slow over a scale length comparable to physical dimensions of the wave packet describing the motion of the carrier.

In the steady state case :

$$\frac{\partial f}{\partial t} = 0 \,. \tag{3.22}$$

In this case BTE is approximated to the *relaxation-time* form:

$$\frac{df}{dt} = -\frac{(f-f_o)}{\tau} \,, \tag{3.23}$$

where f_o is the quasi thermal equilibrium distribution function and τ is the relaxation time defined as [32] :

$$\tau = \tau_p \left[\frac{E_e}{k_B T} \right]^r , \tag{3.24}$$

where the exponent r varies between $-1/2$ and $3/2$, τ_p is a proportionality factor, E_e is the electron energy, and k_B is Boltzmann's constant.

The first order solution of equation (3.23) is :

$$f = f_o \pm \tau \frac{q}{k_B T} . f_o F.u \,, \tag{3.25}$$

where F is the external force. From this solution the electron/hole current densities could be written as :

$$J_n = -q \int u.f d^3k = q\mu_n n F_n \tag{3.26}$$

$$J_p = +q \int u.f d^3k = q\mu_p p F_p \,, \tag{3.27}$$

where μ is the mobility defined as :

$$\mu = \frac{q}{m^*} <\tau> . \tag{3.28}$$

In this equation m^* is the particle's effective mass, and $<\tau>$ is the average relaxation time defined for an isotropic Fermi-Dirac function and a simple band structure as [5] :

$$<\tau> = \frac{-2}{3} \frac{\int_0^\infty \frac{\partial f_o}{\partial \xi} \xi^{3/2} \, d\xi}{\int_0^\infty f_o \xi^{1/2} \, d\xi} \,, \tag{3.29}$$

where

$$\xi = \frac{E_e}{k_B T} . \tag{3.30}$$

Therefore, we reach the following forms describing the continuity equations :

$$J_n = q\mu_n n E + q D_n \nabla n \,, \tag{3.31}$$

$$J_p = q\mu_p p E - q D_p \nabla p \,, \tag{3.32}$$

The first term of equations (3.31) and (3.32) represents the drift component of the current, the second term represents the diffusion component. The diffusion coefficients D are formulated according Einstein relations as :

$$D_{n,p} = \mu_{n,p}\left[\frac{k_B T}{q}\right], \tag{3.33}$$

where T is the temperature.

3.1.4. Current Density Equations in a Magnetic Field :

When a semiconductor device is exposed to a magnetic field, the moving charge carriers are subject to the Lorentz force :

$$\vec{F} = q\,\vec{v} \times \vec{B}, \tag{3.34}$$

where $\vec{v}$ is the drift velocity, $\vec{B}$ is the magnetic induction vector and q is the charge of the carrier. The Lorentz force causes the deflection of charge carriers creating the Hall effect, magnetoconcentration and magnetoresistance. These phenomena are exploited in the design of magnetic sensors.

Equations (3.31), (3.32) are modified within a magnetic field as follows :

$$\vec{J}_n = (q\mu_n n\vec{E} + qD_n\vec{\nabla}n) - \mu_n^*(\vec{J}_n \times \vec{B}), \tag{3.35}$$

$$\vec{J}_p = (\,q\mu_p p\vec{E} + qD_p\vec{\nabla}p\,) + \mu_p^*(\vec{J}_p \times \vec{B}), \tag{3.36}$$

where $\mu_{n,p}$ is the drift mobility, and $\mu_{n,p}^*$ is the Hall mobility. These two mobilities are related as follows :

$$\mu_{n,p}^* = r_H \mu_{n,p}\,. \tag{3.37}$$

The Hall ratio r_H is determined by :

$$r_H = \frac{<\tau^2>}{<\tau>^2}\,. \tag{3.38}$$

To summarize, the behavior of a semiconductor device is described by the following set of partial differential equations :

$$\nabla^2\phi = \frac{q}{\varepsilon}(n - p + N_D - N_A) \tag{3.15}$$

$$\nabla\vec{J}_n - q\frac{\partial n}{\partial t} = qR \tag{3.19}$$

$$\nabla\vec{J}_p + q\frac{\partial p}{\partial t} = -qR \tag{3.20}$$

$$\vec{J}_n = (q\mu_n n\vec{E} + qD_n\vec{\nabla} n) - \mu_n^*(\vec{J}_n\times\vec{B})\ , \tag{3.35}$$

$$\vec{J}_p = (\ q\mu_p p\vec{E} + qD_p\vec{\nabla} p\ ) + \mu_p^*(\vec{J}_p\times\vec{B})\ . \tag{3.36}$$

The above equations are the models discussed in section (2.2). The simulation of a semiconductor device requires solving these equations according to the steps mentioned in section (2.3).

3.2 Models of the Physical Parameters

In order to proceed with the simulation of a semiconductor device according to SDBE models of the previous section, the parameters of these models must be quantified. These parameters are the previously mentioned, physical parameters. They could be either explicitly or implicitly stated in the model. An example of an explicit physical parameter is the carrier mobility, an implicit is bandgap narrowing.

In the remainder of this chapter we discuss these parameters, their models and mechanisms. The parameters are, doping profile, carrier mobility, carrier generation and recombination, bandgap narrowing, dopant species and the boundary conditions. We wish to note that, doping profiles are usually part of a process simulator rather than a device simulator. In this work we have included such an aspect based on realistic, proven and widely accepted models. Our purpose was to minimize Albertina's dependence on external packages.

3.3. Doping Profile :

The introduction of dopants to alter the conductivity of a semiconductor is achieved through two techniques, namely diffusion and ion implantation. The techniques are modeled as :

1. Diffusion : In this technique dopant species over a wide range of concentrations are introduced into the silicon wafer surface and diffuse in accordance to Fick's two laws:

   $$J_d = -D \frac{\partial C(x,t)}{\partial x} , \qquad (3.37)$$

   $$\frac{\partial C(x,t)}{\partial t} = D \frac{\partial^2 C(x,t)}{\partial x^2} , \qquad (3.38)$$

   where J_d is the diffusion flux, C(x,t) is the solute concentration as a function of space x, and time t, with D being the diffusion constant.

2. Ion Implantation : In this technique a beam of accelerated ions of the dopant species is applied to the surface of a the semiconductor surface. As a result, the dopant will penetrate into the substrate beyond the surface. The dopant distribution within the substrate depends on certain physical properties of both the dopant and the target.

In this work we consider only the distribution obtained by ion implantation. The following is a discussion of the parameters and models of doping profiles.

3.3.1. Distribution Parameters :

An implanted ion encounters scattering events due to collisions with electrons and atoms during its travel within the target. Consequently, the ion energy is reduced until it comes to rest.

The total path length of the ion is defined as the range R. This path is projected in the following two directions :

1. Parallel to the incidence direction hence called "Projected Range" R_p .
2. Normal to the incidence direction called "Standard Deviation" ΔR_p.

The dopant distribution function within the substrate could be described as [33] :

$$F(x) = \int_{-\infty}^{\infty} x\, f(x) dx \, , \tag{3.39}$$

where $f(x)$ is the frequency function and satisfies the condition :

$$\int_{-\infty}^{\infty} f(x) dx = 1 \, . \tag{3.40}$$

Accordingly, R_p and ΔR_p are redefined in terms of the moments of the distribution function as follows. The first moment is :

$$u_1 = R_p = \int_{-\infty}^{\infty} x\, f(x)\, dx \, . \tag{3.41}$$

Three higher order moments are defined as :

$$u_i = \int_{-\infty}^{\infty} (\, x - R_p\,)^i f(x)\, dx \;\ldots\ldots\; i = 2, 3, 4 \, . \tag{3.42}$$

The dopant distribution parameters are defined in terms of the distribution moments as follows :

$$\Delta R_p = \sqrt{u_2} \tag{3.43}$$

and the definitions :

$$skewness = \gamma = \frac{u_3}{\Delta R_p^3} \tag{3.44}$$

$$kurtosis = \beta = \frac{u_4}{\Delta R_p^4} \, . \tag{3.45}$$

3.3.2. Calculation of the Distribution Parameters :

Methods of calculating R_p and ΔR_p have been discussed and formulated in several references [34, 35]. The calculation of the distribution function moments are computationly expensive and cumbersome. Therefore, we selected an easier approach which is a simple polynomial fit [36]. According to this approach, the projected range and the standard deviation are calculated as follows :

$$R_p = \sum_{j=0}^{j=n} a_j E_I^j \tag{3.46}$$

$$\Delta R_p = \sum_{j=0}^{j=n} b_j E_I^j , \tag{3.47}$$

where E_I is the Implantation Energy in KeV. The coefficients a_j and b_j are dopant dependent and are tabulated for Silicon as follows [36] :

Coefficient	Boron	Phosphorus	Antimony	Arsenic
a_1	3.338×10^{-3}	1.259×10^{-3}	8.887×10^{-4}	9.818×10^{-4}
a_2	-3.308×10^{-6}	-2.743×10^{-7}	-1.0138×10^{-5}	-1.022×10^{-5}
a_3	0	1.290×10^{-9}	8.372×10^{-8}	9.067×10^{-8}
a_4	0	0	-3.056×10^{-10}	-3.442×10^{-10}
a_5	0	0	4.028×10^{-13}	4.608×10^{-13}

Table (3.1) : Coefficients of R_p polynomial fit.

Coefficient	Boron	Phosphorus	Antimony	Arsenic
b_1	1.781×10^{-3}	6.542×10^{-4}	2.674×10^{-4}	3.652×10^{-4}
b_2	-2.086×10^{-5}	-3.161×10^{-6}	-2.885×10^{-6}	-3.820×10^{-6}
b_3	1.403×10^{-7}	1.37×10^{-8}	2.311×10^{-8}	3.235×10^{-8}
b_4	-4.545×10^{-10}	-2.252×10^{-11}	-8.310×10^{-11}	-1.202×10^{-10}
b_5	5.525×10^{-13}	0	1.084×10^{-13}	1.601×10^{-13}

Table (3.2) : Coefficients of ΔR_p polynomial fit.

Plots of these parameters according to equations (3.46) and (3.47) are provided in appendix III. Albertina's graphics package was used for these plots. Implementation of this package is described in chapter 6.

3.3.3. Distribution Functions (Doping Profiles) :

The dopant distribution (doping profile) in the target substrate is approximated by several distribution functions. The functions are classified according to the number of Cartesian coordinates it encompasses. These functions are given in the following sections :

3.3.3.1. One Dimensional Distribution Functions :

1. One Dimensional Gaussian distribution : This is most widely used to model the dopant distribution. The result is centered around the projected range R_p. The dopant concentration $C(x)$ at a distance x normal to the substrate surface is calculated as follows :

$$C(x) = \frac{N}{\sqrt{2\pi}\Delta R_p}\exp\left[-\frac{(x-R_p)^2}{2\Delta R_p^2}\right], \tag{3.48}$$

where N is the implanted dose. Plots of this distribution function are given in figures (3.3) and (3.4) boron in silicon at different implantation energies.

2. The Pearson distribution function : The Pearson distribution function $f(x)$ is defined by :

$$\frac{df(x)}{dx} = \frac{(x-a)\,f(x)}{b_2x^2 + b_1x + b_0} . \tag{3.49}$$

Considering the above definitions of R_p, ΔR_p, β and γ, equation (3.49) has a general solution of the form :

$$f(x) = K \left[b_2(x-R_p)^2 + b_1(x-R_p) + b_0\right]^{\frac{1}{2b_1}} .$$

$$\exp\left[-\frac{\frac{b_1}{b_2} + 2a}{\sqrt{4b_2b_0 - b_1^2}} arctan \frac{2b_2(x-R_p) + b_1}{\sqrt{4b_2b_0 - b_1^2}}\right] , \tag{3.50}$$

where K is a constant and the constants a, b_0, b_1 and b_2 are given by :

$$a = \frac{\Delta R_p\ \gamma(\beta + 3)}{A_p} , \tag{3.51}$$

$$b_0 = -\frac{\Delta R_p^2(4\beta - 3\gamma^2)}{A_p} , \tag{3.52}$$

$$b_1 = a , \tag{3.53}$$

$$b_2 = -\frac{(2\beta - 3\gamma^2 - 6)}{A_p} , \tag{3.54}$$

with

$$A_p = 10\beta - 12\gamma^2 - 18 . \tag{3.55}$$

There are various types of Pearson distribution function. The types range from Pearson I - Pearson VII, depends on the value of the kurtosis parameter β_p. Hofker [37], states that a Pearson IV type distribution is well suited to match boron profiles in silicon. Ryssel et al. [33] claimed it true for boron and arsenic as dopants in SiO_2 and Si_3N_4 targets.

The Pearson function is of type IV when the kurtosis β_p is :

$$\beta_p \geq \beta_{\min} = \frac{48 + 39\gamma^2 + 6(\gamma^2 + 4)^{3/2}}{32 - \gamma^2} , \tag{3.56}$$

Ryssel [33] found that arsenic distribution can be fitted excellently at $\beta_p = \beta_{\min}$ and for boron profiles at $\beta_p = 1.5\ \beta_{\min}$.

3.3.3.2. Two Dimensional Distribution Functions :

This category of distribution functions provide a mathematical model for the distribution in two directions:

1. X-direction normal to the device surface into the device bulk.
2. Y-direction parallel to the surface.

In Albertina we have provided a model known as, *Two Dimensional Gaussian Distribution*. This model was proposed by Furukawa et al. [38]. In this model *Rsubp*, ΔR_p and ΔR_l, the projected range, vertical and lateral standard deviation respectively, are described in integral formulas. The formulas are derived from the equations proposed by Lindhard, Schraff, and Schiφtt, known as the LSS theory [39]. According to this theory the distance travelled by an ion into a target is considered as a probability described by an integro-differential equation. This equation uses differential scattering cross-section to describe the interaction between the ion and the target atom. The resulting equations can be numerically solved using the Runge-Kutta-Gill method. The solution is proposed for a Gaussian distribution within a range of moments. The model is seen as the superposition of many fine beams of the incident ions through a mask window of width 2a, and is formulated as follows:

$$C(x,y) = \frac{N}{\sqrt{2\pi}R_p^2}.\exp\left[\frac{(x - R_p)^2}{2R_p^2}\right].\ erfc\left[\frac{(a - y)}{\sqrt{2}.\Delta R_l}\right] , \tag{3.57}$$

where 2a is the mask width, N is the implanted dose and,

$$erfc(t) = \int_t^{\infty} \exp(-u^2)\, du \, . \tag{3.58}$$

Other methods of calculating two dimensional distribution functions were proposed by Hobler et al. [40] and Giles [41], but where not included in Albertina as they are tentative.

3.4. Carrier Mobility :

On reviewing the "Carrier Transport Equations" in section (3.1.2) the term "Relaxation time, τ" has been introduced. Where τ is defined as the rate at which electrons and holes are caused to change their momentum vector **k** due to collisions. Consequently the term carrier mobility μ, which is intuitively easier to imagine, has been defined according to the following relation :

$$\mu = \frac{q}{m^*} \langle\tau\rangle \, . \tag{3.59}$$

Boltzmann Transport Equation (3.21) describes the motion of carriers in terms of rate of particle collision. The moving particles collide with a number of entities, e.g. other carriers, phonons. Collisions occur according to several mechanisms, known as the *scattering mechanisms*. Therefore, modeling the carrier mobility parameter μ has to account for the various scattering mechanisms. The theoretical definition of μ in equation (3.59) has encompassed most scattering mechanisms. The problem with this model is the definition of τ as a function of the wave vector **k** requires an advanced wave-mechanical treatment involving the quantum theory of collision processes. Since the resulting models of too complex for device simulation, empirical models are used instead. In this sub-section we present the important scattering mechanisms that affect charge carriers in semiconductor devices. We then proceed to present the empirical models of the carrier mobility that are used by Albertina.

3.4.1. Scattering Mechanisms and Theoretical Mobility Models :

In non-polar semiconductors such as germanium and silicon the scattering mechanisms that prevail are the following :

1. Deformation potential acoustic phonons : It is the fundamental process by which carriers are scattered due to the thermal vibrations of the lattice. The resulting mobilities are temperature dependent. A theoretical derivation of a model for this mechanism leads to [7] :

$$\mu_n^{ac} = \frac{2\sqrt{2}\pi q \hbar^4 C}{3(m_n^*)^{5/2} E_{ac}^2 (k_B T)^{3/2}} , \tag{3.60}$$

$$\mu_p^{ac} = \frac{2\sqrt{2}\pi q \hbar^4 C}{3(m_p^*)^{5/2} E_{av}^2 (k_B T)^{3/2}} , \tag{3.61}$$

where $C = 10^5 VAscm^{-3}$ is the average longitudinal elastic constant of the material, E_{ac} and E_{av} are the deformation potential constants of the conduction and valence bands, m_n^*, m_p^* are the electron and hole effective masses.

2. Optical mode scattering : This scattering process is caused by longitudinal optical phonons and is given by [42] :

$$\frac{1}{\mu_{op}} = \frac{2\pi^{3/2}\alpha_{op} m_e \nu_e}{2q} [f(z)]^{-1} , \tag{3.62}$$

$$f(z) = \eta(z) \frac{[e^z - 1]}{\sqrt{z}} , \tag{3.63}$$

$$z = \frac{\hbar \nu_e}{k_B T} , \tag{3.64}$$

where α_{op} is the polar constant, ν_e is the phonon's frequency and $\eta(z)$ is a slowly varying function. Due to the exponential factor in $f(z)$, $\frac{1}{\mu_{op}}$ becomes very small at low temperatures. For non-polar crystals the value of α_{op} is small, therefore the effect of this mechanism is negligible. For polar crystals (e.g. GaAs) α is close to unity, and should therefore be considered.

3. Intervalley Scattering : This mechanism occurs with semiconductors having at least two different minima in the conduction band at $\mathbf{k}_1$ and $\mathbf{k}_2$. An electron with wave vector $\mathbf{k}$ close in value to $\mathbf{k}_1$ may change its wave-vector to a value $\mathbf{k}'$ nearly equal to the second minimum $\mathbf{k}_2$ and consequently emit or absorb a phonon. This mechanism does take place at low temperatures. Theoretical models for such mechanism are limited to cases of $k_BT \gg \hbar\omega_o$ and $k_BT \ll \hbar\omega_o$ where $\hbar\omega_o$ is the phonon energy.

4. Ionized Impurity Scattering : In a classical treatment of the collision of electrons and ionized impurity ions in a Coulomb field, Conwell and Weisskopf [43] have derived a theoretical mobility model. The model was developed as a result of investigating the resistivity of germanium according to the following :

$$\mu_i = G.\left[\ln(x)\right]^{-1}, \qquad (3.65)$$

where

$$G = \frac{64\sqrt{\pi}\varepsilon^2(2k_BT)^{3/2}}{N_iZ^2q^3\sqrt{m^*}}, \qquad (3.66)$$

$$x = 1 + \left[\frac{12\pi\varepsilon k_BT}{Zq^2N_i^{1/3}}\right]^2, \qquad (3.67)$$

where Z is the number of valence electrons. Brooks [44], has later modified x as :

$$x = \frac{24\varepsilon m^*(k_BT)^2}{q^2\hbar^2N_i}. \qquad (3.68)$$

5. Surface Scattering : This scattering mechanism is fundamental to devices where current flow occurs along the surface, i.e. MOS devices. It is explained in terms of quantized motion of carriers, and the splitting of the conduction band into

sub-bands. Experimental models can be interpreted only at low temperatures. The complexity of the mechanism allows only empirical models that coincide with experimental findings. Two of empirical surface scattering models will be discussed in the following section.

As a summary, all theoretical models of carrier mobility has been derived considering only one specific scattering mechanism. Modeling the interaction of different mechanisms is difficult. The validity of combining these models according to the Mathiessen rule, viz.

$$\mu_o^{-1} = \mu_{ac}^{-1} + \mu_{op}^{-1} + \mu_i^{-1} , \tag{3.69}$$

has been subject to debates. Its application is not valid in certain cases, as the total relaxation time is a non-linear combination of individual relaxation times, i.e. :

$$<\tau> = < \left(\sum_{i=0}^{i=j} \tau_i^{-1} \right)^{-1} > \neq \left(\sum_{i=0}^{i=j} <\tau_j>^{-1} \right)^{-1} . \tag{3.70}$$

Where the operator < > represents the average over the energy distribution. For simulation purposes various empirical mobility models have been developed incorporating different scattering mechanisms. These models are used in Albertina's library.

Finally, other scattering mechanisms do exist, such as carrier-carrier, neutral impurity, and dislocation scattering. These mechanisms are not of vital importance to semiconductor devices. Discussions of those mechanisms and their formulations could be found in several books [6, 32, 42].

3.4.2. Empirical Mobility Models :

According to the above discussion, we have selected the following empirical mobility models that encompass the important scattering mechanisms relevant to semiconductor devices. These models are named after their developers and are as follows :

1. Caughey and Thomas:

In [45], Caughey and Thomas presented two empirical models for the mobility. The first is as a function of doping concentration, the second accounts for the electric field. The models are developed by parameter fitting techniques as follows :

a. Doping Concentration dependency model :

$$\mu^{ctd} = \frac{\mu_{\max} - \mu_{\min}}{1 + \left[\frac{N}{N_{ref}}\right]^{\alpha_c}} + \mu_{\min}\,, \tag{3.71}$$

where μ^{ctd} is the mobility, N is the doping concentration, $\mu_{\max}$, $\mu_{\min}$, α_c and N_{ref} are empirical fitting parameters. Values for the parameters are provided in appendix II.

b. Electric Field dependency model :

$$\mu^{cte} = \frac{\mu^{ctd}}{\left[1+(\frac{E}{E_c})^{\beta_c}\right]^{\frac{1}{\beta_c}}}\,, \tag{3.72}$$

where E is the electric field in V/cm. The term $\mu^{ctd} = \mu(E = 0)$ is calculated according to equation (3.71). Values for the empirical parameters E_c and β_c are provided in appendix II.

This model is widely used as it encompasses both lattice and impurities scattering mechanisms.

2. Thornber and Gummel : This model has been originally developed by Scharfetter and Gummel [46]. It accounts for the variation of the carrier mobility with the electric field. This effect is based on the relation of the average drift velocity v_d as the product of the electric field component parallel to the current flow $E_{\|}$ times μ_o the carrier mobility [42],

$$v_d = \mu_o . E_{\parallel} . \qquad (3.73)$$

For a certain range of accelerating electric field $E_{\parallel}$, v_d is unaltered from the saturation velocity v^{sat}. The result is a reduction in the effective carrier mobility.

The Scharfetter and Gummel model has been modified to comply with Thornber's scaling rules [47]. In this paper Thornber discusses scaling the scattering rate of the BTE by a factor $\Gamma_T > 1$. Under space-time invariant conditions this scaling will reduce the drift velocity. Carrier mobility is also reduced by a factor of $\frac{1}{\Gamma_T}$ while v^{sat} is unaltered. Thornber scaled the carrier momentum by a factor $\gamma_t > 0$. This scaling renders an invariant mobility and alters the saturation velocity. Thus, Γ_T, γ_T scaling factors are independent and complementary. The scaled model, hence known as Thornber-Gummel is :

$$\mu^{tg} = \frac{\mu^{ctd}}{\left[1 + (\frac{\mu^{ctd}E}{u_l})^2 \, (G + \frac{\mu^{ctd}E}{u_l})^{-1} + (\frac{\mu^{ctd}E}{v^{sat}})^2 \right]^{1/2}} , \qquad (3.74)$$

where , u_l is the longitudinal acoustic phonon velocity, G is a fitting parameter, μ^{ctd} is the initial mobility value calculated according to equation (3.71). The temperature dependence of the drift velocity saturation v^{sat} is calculated according to [48] :

$$v_n^{sat} = 10^7 \, (\frac{T}{300})^{-0.87} , \qquad (3.75)$$

$$v_p^{sat} = 8.37.10^6 \, (\frac{T}{300})^{-0.52} . \qquad (3.76)$$

This model incorporates lattice and impurity scattering with the effect of drift velocity saturation for high electric fields.

3. Scharfetter and Gummel :

Two of the models used in Albertina have been developed in [45]. The first accounts for ionized impurity scattering the second accounts for the effect of the electric field. These models are:

a. For ionized impurity scattering this model is a function of the doping concentration and is valid at 300K :

$$\mu_{n,p}^{sgd} = \frac{\mu_o}{\left[1 + \dfrac{N}{N_{ref} + \dfrac{N}{S_{n,p}}}\right]^{\frac{1}{2}}} . \tag{3.77}$$

Values of the fitting parameters μ_o, N_{ref} and $S_{n,p}$ are tabulated in appendix II.

b. The second model includes the effect of the electric field and the ionized impurity density :

$$\left[\frac{\mu^{sgd}}{\mu^{sgi}}\right]^2 = 1 + \left[\frac{N_D}{\dfrac{N_D}{S} + N}\right] + \left[\frac{(\dfrac{E}{A})^2}{\dfrac{E}{A} + F}\right] + (\frac{E}{B})^2 , \tag{3.78}$$

where S, N, A, F and B are constants listed in appendix II, N_D is the doping concentration and E is the electric field. The parameter μ^{sgi} the initial mobility calculated from the previous model (3.77).

4. Arora's temperature dependence model : Arora et al. [49], have developed a model that is analogous in structure to the Caughey and Thomas models. It accounts for temperature variation. The author claims accuracy within 13% in the temperature range of 250 - 500 K and within a total ionized impurity range of 10^{13} to $10^{20} cm^{-3}$. This model accounts for lattice and impurity scattering mechanism, and reads :

$$\mu_{n,p}^{a} = a_{n,p}(\frac{T}{300})^{\alpha_n} + \frac{\left[b_{n,p}(\dfrac{T}{300})^{\beta_n}\right]}{\left[1 + \dfrac{N}{c_{n,p}(\dfrac{T}{300})^{\gamma_n}}\right]} , \tag{3.79}$$

where $\mu^{a}_{n,p}$ is the mobility for electron and holes respectively, N is the impurity concentration, $a_{n,p}$, α_a, $b_{n,p}$, β_a, $c_{n,p}$ and γ_a are coefficients listed in appendix II, and T (Kelvin) is the temperature.

5. Yamaguchi's surface scattering model : This is the first of two empirical models describing surface scattering included in Albertina's library of physical parameters. K. Yamaguchi published his model and later modified it in [50] according to Thornber's scaling rule. The model is based on the Scharfetter and Gummel's (3.78) empirical expression. It has considered the surface quantization and surface roughness with a dependence on the gate field in MOS devices. Hence, the author derived an empirical reduction expression fitting experimental results of the form :

$$g\,(E_t)=\left[1+\alpha_y E_t\right]^{-1/2}, \tag{3.80}$$

where E_t is the transverse electric field at the gate surface and α_y is a fitting coefficient tabulated in appendix II. The model finally reads :

$$\mu^{y}(E_t)=\mu^{tg}\,g(E_t)\,, \tag{3.81}$$

where μ^{tg} is the mobility calculated in accordance to the Thornber-Gummel model equation (3.74). This model expresses surface scattering in addition to the other mechanisms included in the Thornber-Gummel model.

6. Selberherr's surface scattering model :

An alternative formula for the surface scattering mechanism has been suggested by Selberherr [6], and is of the form:

$$\mu^{s}_{n,p}=\mu^{tg}_{n,p}+\frac{x+x^{ref}_{n,p}}{x+b_{n,p}x^{ref}_{n,p}}\,, \tag{3.82}$$

where x is the distance perpendicular to the surface, x^{ref} is a characteristic length describing the range of surface scattering influence, defined as :

$$x^{ref}_{n,p} = \frac{x^0_{n,p}}{1+\dfrac{E_{n,p}}{E^{crit}_{n,p}}} , \qquad (3.83)$$

where $x^0_{n,p}$ and $E^{crit}_{n,p}$ are constants defined in appendix II. Here b describes the strength of the influence of this mechanism expressed in terms of E_t the electric field component normal to current flow as follows :

$$b_n = 2 + \frac{E_t}{E^{crit}_{tn}} , \qquad (3.84)$$

$$b_p = 2 + \frac{E_t}{E^{crit}_{tp}} , \qquad (3.85)$$

where E^{crit}_{tn} and E^{crit}_{tp} are constants listed in appendix II, and $\mu^{tg}_{n,p}$ is the initial mobility value obtained from the Thornber-Gummel model.

3.5. Carrier Generation and Recombination :

The decomposition of equation (3.18) into (3.19) and (3.20) has introduced the term R representing carrier generation and recombination per unit volume. Such a process is a result of different physical mechanisms, such as photon emission or absorption, phonon emission and absorption, three particle Auger impact transitions, trap assisted, impact ionization and other mechanisms that are not of the same importance with respect to semiconductors (e.g. plasmon interaction .. etc).

Radiative transitions of photons are the primary mechanisms causing generation and recombination in "direct gap" semiconductors, yet, for "indirect gap" semiconductors such as Ge and Si, this phenomenon occurs mainly via trap centers. Therefore, in Albertina we have incorporated the following mechanisms and models :

1. Shockley-Read-Hall (SRH) mechanism and model : The SRH theory of trap assisted recombination was developed in 1952 [51, 52]. It explained the processes in terms of electrons and holes moving from/to either the conduction or

valence band via a defect created trap within the following four steps :

a. Electron capture : An electron's transition from the conduction band to a trap state.

b. Electron emission : A previously trapped electron moves to the conduction band, and thus freeing the trap state.

c. Hole emission : An electron from the valence band is trapped, thus creating a hole in the valence band and occupying a trap state.

d. Hole capture : A trapped electron moves to the valence band, neutralizing a hole and freeing a trap state.

After several derivation steps and assumptions the SRH model for the net generation-recombination model reads :

$$(R - G)_{SRH} = \frac{(np - n_{ie}^2)}{\tau_{po}(n+n_1) + \tau_{no}(p+p_1)} , \tag{3.86}$$

where τ_{no} and τ_{po} are electron and hole lifetimes respectively. They are defined as [53, 54] :

$$\tau_{no,po} = \frac{\tau_{n,p}^o}{1+\dfrac{N_i}{N_{n,p}^o}} , \tag{3.87}$$

where $\tau_{n,p}^o$, $N_{n,p}^o$ are constants listed in appendix II, and N_i is the net impurity concentration. The variables n_1, p_1 are the electron and hole concentration in the conduction band when the Fermi level coincides with the energy level of the single-level trap. Since this is usually unknown, the most effective trap level is assumed [55] :

$$n_1 = p_1 = n_{ie} , \tag{3.88}$$

where

$$n_{ie}^2 = n_o p_o := n_i^2 \exp\left[\frac{\Delta E_G}{k_B T}\right] , \tag{3.89}$$

where ΔE_G is the bandgap narrowing effect.

2. Auger mechanism and model : Debates regarding the validity of this mechanism have risen as it is only qualitatively understood. Nevertheless, [42] quotes its importance for semiconductors at high temperatures and recommends combining this model with radiative recombination. The Auger model describes the recombination process as a one step disposition of energy to a carrier thus causing its direct transition from one band to another. This mechanism is also known as "band-to-band radiation". It takes place in the following steps :

   a. Electron capture : An electron from the conduction band moves to the valence band, transmitting the energy difference to another electron in the conduction band, and in the valence band it combines with a hole.

   b. Hole capture : An electron from the conduction band moves to the valence band transferring the energy difference to a hole in the valence band, the hole moves away from the valence band edge, and the electron combines with a hole in the valence band.

   c. Electron emission : An electron in the valence band consumes the energy of a highly energized electron allowing it to move to the conduction band and creating a hole in the valence band.

   d. Hole emission : A highly energized hole in the valence band transmits its energy to an electron allowing it to move to the conduction band, creating a hole in the valence band.

   The Auger model for the net generation and recombination is of the form :

$$(R - G)_A = (c_n^A \, n + c_p^A \, p)(np - n_{ie}^2) , \tag{3.90}$$

where $c_n^A = 2.8 \, X \, 10^{-31} \, cm^6 s^{-1}$, $c_p^A = 9.9 \, X \, 10^{-32} \, cm^6 s^{-1}$, are known as the Auger capture coefficients, n and p are electron and hole concentrations respectively, and n_{ie} is the effective intrinsic carrier concentration.

3. Impact ionization: This pure generation mechanism could be viewed as the inverse of the Auger process with the main difference being in the energy sources. The Auger mechanism is stipulated under equilibrium. Impact ionization is stipulated under non-equilibrium conditions of large electric fields. The result of this mechanism is to heat up the carriers, leading to Zener effect or Avalanche breakdown.

The process is conceived in the two steps namely "electron and hole emission" as described in the partial steps of the Auger mechanism, and is formulated as :

$$G_{II} = \frac{1}{q}\left(\alpha_n^i \left|J_n\right| + \alpha_p^i \left|J_p\right|\right) , \tag{3.91}$$

or

$$G_{II} = \alpha_n^i n v_n + \alpha_p^i p v_p , \tag{3.92}$$

where J_n, J_p are the electron and hole current densities. In terms of the electron and hole drift velocities v_n, v_p the ionization coefficients α_n^i and α_p^i are defined as the reciprocals of the average distance travelled by an electron or hole between consecutive ionizations. They are suggested by Okuto and Crowell [56] in the following empirical form :

$$\alpha_{n,p}^i = a_{n,p} \exp\left[-\left(\frac{b_{n,p}}{E_{\|}}\right)^2\right] , \tag{3.93}$$

where $E_{\|}$ is the electric field component in the direction of the current flow. The variables a and b are functions of temperature :

$$a_n = 0.426 \; (\; 1 + 3.0510^{-4} \; (T - 300)) \; , \tag{3.94}$$

$$b_n = 4.81\times10^5 \; (\; 1 + 6.86 \; 10^{-4} \; (T - 300)) \; , \tag{3.95}$$

$$a_p = 0.243 \; (\; 1 + 5.35 \; 10^{-4} \; (T - 300)) \; , \tag{3.96}$$

$$b_p = 6.53\times10^5 \; (\; 1 + 5.87 \; 10^{-5} \; (T - 300)) \; . \tag{3.97}$$

Finally, the total recombination rate is a result of the sum of the models of the different mechanisms involved i.e. :

$$(R-G) = (R-G)_{SRH} + (R-G)_A - G_{II}\,. \qquad (3.98)$$

3.6. Band Gap Narrowing :

This phenomenon occurs at large impurity concentrations in semiconductors. It has been attributed to the alteration of the density of energy state to a non-parabolic distribution, as an impurity band is formed as well as band tails on the edges of the conduction and valence bands.

Band gap narrowing occurs at doping concentrations greater than $10^{17} cm^{-3}$ and strongly affects the behavior of silicon devices, particularly the minority carrier in the highly doped regions. This is demonstrated in the following relations between the electron and hole concentrations n and p vs. the effective intrinsic carrier concentration n_{ie} which in turn could be expressed as a function of the bandgap narrowing ΔE_G as follows :

$$n = n_{ie} \exp\left[\frac{q(\phi - \phi_n)}{k_B T}\right], \qquad (3.99)$$

$$p = n_{ie} \exp\left[\frac{q(\phi_p - \phi)}{k_B T}\right], \qquad (3.100)$$

$$n_{ie}^2 = n_i^2 \exp\left(\frac{\Delta E_G}{k_B T}\right), \qquad (3.101)$$

where ϕ_n, ϕ_p are electron and hole quasi-Fermi potentials, ϕ is the electro-static potential.

In Albertina we have incorporated the following bandgap narrowing models:

1. Slotboom and De Graaf model :

In [57], an empirical model for bandgap narrowing has been developed by fitting experimental data obtained from measurements of I_C vs. V_{EB} of NPN bipolar transistors. Hence the authors determined ΔE_G in the base region, within a doping concentration range of $4.10^{15} - 2.5\ 10^{19}\ cm^{-3}$, and reads :

$$\Delta E_G = V_1 \left[\ln(\frac{N}{N_o}) + \sqrt{(\ln^2 \frac{N}{N_o} + C)} \right] , \qquad (3.102)$$

where $V_1 = 9\ mV$, $N_o = 10^{17} cm^{-3}$, $C = 0.5$ and N is the impurity concentration.

2. Van Overstraeten et al. model : In [58], the authors calculated the capacitance of a linearly graded heavy doped junction [10^{19}cm^{-3}]. In equations (3.99), (3.100) for an applied voltage V_a, the following relation was used :

$$\phi_p = -\phi_n = \frac{V_a}{2} . \qquad (3.103)$$

As a result Van Overstraeten et al. have derived the following expression for bandgap narrowing :

$$\Delta E_G = \frac{k_B T}{q} \ln \frac{n_{ie}^2}{n_i^2} , \qquad (3.104)$$

where n_i is the intrinsic carrier concentration, and the effective carrier concentration $n_{ie} = N_D - N_A$. Compared to experimental results plotted on C^{-3} vs. V graphs, the model showed considerable agreement.

3. Thurmond model : Experimental results show that for most semiconductors bandgap variations are inversely proportional to temperature. Thurmond [59] developed a universal expression for bandgap variation as a function of temperature :

$$E_g(T) = E_g(0) - \frac{\alpha_t T^2}{(T + \beta_t)} , \qquad (3.105)$$

where values for $E_g(0)$, α_t and β_t for different materials are listed in the following table :

Material	$E_g(0)$	$\alpha_t \times 10^{-4}$	β_t
GaAs	1.519	5.405	204
Si	1.170	4.73	636
Ge	0.7437	4.774	235

Table (3.3) : Coefficients of the Thurmond model.

3.7. Dopant :

As R_p and ΔR_p are calculated using the polynomials of equations (3.46) (3.47), it is important to define which donor or acceptor species the region is doped with. This is done in order to locate the coefficients of the polynomials as per tables (3.1, 3.2). In our system the recognized dopants are, phosphorus, boron, antimony and arsenic.

3.8. Boundary Conditions :

The basic semiconductor equations are posed in a bounded domain representing the device geometry. Although the device is a three-dimensional structure, the inherent symmetry of the problem allows the reduction of its space dimensions to two [6]. Therefore, suitable boundary conditions must be applied to contacts, interfaces and external surfaces so that the device operating conditions are properly simulated. The two boundary conditions that are considered here are Dirichlet and von Neumann :

1. The Dirichlet boundary condition is defined for the electrostatic potential $\phi(t)$ by :

$$\phi(t) - \phi_b - \phi_D(t) = 0 \,, \tag{3.106}$$

where $\phi_D(t)$ is the externally applied bias. The built-in potential ϕ_b is given by :

$$\phi_b = \frac{k_B T}{q} \sinh^{-1}\left[\frac{N_D^+ - N_A^-}{2n_i}\right] , \tag{3.107}$$

where N_D^+, N_A^- are singly ionized donors and acceptors respectively.

For an ohmic contact we assume that the conduction is ideal, i.e. no voltage drop occurs over the boundary, reducing equation (3.106) to :

$$\phi(t) - \phi_b = constant \,. \tag{3.108}$$

In other words, for an applied voltage V_a the following equations apply to the

Dirichlet boundary condition :

$$\phi_c = V_a + \frac{k_B T}{q} \sinh^{-1}\left[\frac{N_D^+ - N_A^-}{2n_i}\right], \tag{3.109}$$

$$n_c = \sqrt{\frac{N^2}{4} + n_{ie}^2} + \frac{N}{2}, \tag{3.110}$$

$$p_c = \sqrt{\frac{N^2}{4} + n_{ie}^2} - \frac{N}{2}, \tag{3.111}$$

where

$$N = N_D^+ - N_A^- . \tag{3.112}$$

2. The von Neumann boundary condition deals with the interface between the semiconductor and an insulating material. At such an interface the law of Gauss,

$$\varepsilon_s \frac{\partial \phi_s}{\partial \vec{n}} - \varepsilon_i \frac{\partial \phi_i}{\partial \vec{n}} = Q_f, \tag{3.113}$$

applies, where ε_s and ε_i are the permittivity in the semiconductor and the insulator respectively, with Q_f being the charge at the interface. For thick insulators the field component perpendicular to the interface is assumed zero. Moreover the interface charge is neglected, yielding the von Neumann boundary condition :

$$\frac{\partial \phi}{\partial \vec{n}} = 0 . \tag{3.114}$$

4. The Abstract Models of a User Interface

The problem of Albertina's tediousness and rigidity, resulting in a high susceptibility to errors, has been mentioned in section (2.6). The main goal of this work is to enhance the performance of the package as a whole. This is achieved by restructuring Albertina in order to minimize the effort made by a user throughout a simulation session in terms of :

a. Geometric and physical definition of the simulated object.

b. Invoking and monitoring the solution steps.

One of the main aspects of Albertina's new structure was to provide Albertina with a user interface. According to the discussion in section (2.5), for our application a suitable interface is that of a graphical nature. The interface should behave as a communication medium between the user and the system's main core. The purpose of the user interfaces is to mediate between the user and the system. Mediation reduces the system's complexity and hides certain unnecessary details. In order to design such an interface the intended user and the interface responsibilities have to be defined. The interface should also be flexible to accommodate future expansions.

In this chapter we shall discuss our abstract approach towards restructuring Albertina, defining the main objectives and characteristics of each step. The discussion is conducted in two directions, first from a user point of view and secondly from the view point of the system designer with regard to the new structure. We end the chapter by deriving a correlation between both abstract visions.

4.1. User Perception of Albertina :

The restructuring of Albertina from a user point of view is illustrated in figure (4.1). The intended user is an *expert*, who is familiar with the topic and steps of semiconductor device simulation. From this view the system is composed of three layers.

Each layer is composed of several packages, with each package consisting several modules. Based on this terminology we describe the system as follows :

1. User Interface layer: The first layer is a *"Man-Machine interface "* inserted between the user and system, acting as a mediator between the user and the other layers. The interface allows easy, monitored and flexible communication between the user and the modules of each of the other layers on one hand, and modules of the same layer on the other. Therefore, the interface should be aware of all input and output, and able to relate them to each other.

The user interaction with the system through the interface is graphical in nature. The interface has the following features in accordance with the latest specification of user interfaces :

a. Panels, representing the available operations and tasks available to the user. The panels are used to invoke tasks in accordance with a user defined sequence.

b. Menus of parameters and models for users allocation.

c. Prompts, prompting user for optional or mandatory data.

In terms of operation the following characteristics were kept in mind:

a. Result monitoring and inspection.

b. Initial data modification.

c. Help Messages displayed to the user.

d. Error messages displayed to the user.

e. Multitasking, where time consuming operations (e.g. mesh generator) are executed concurrently with the user interface and/or other operations. The user can then attend to other tasks during the elapsed time of execution.

These features will allow an expert user to set the appropriate parameters and determine the solution strategy.

Albertina Hierarchy

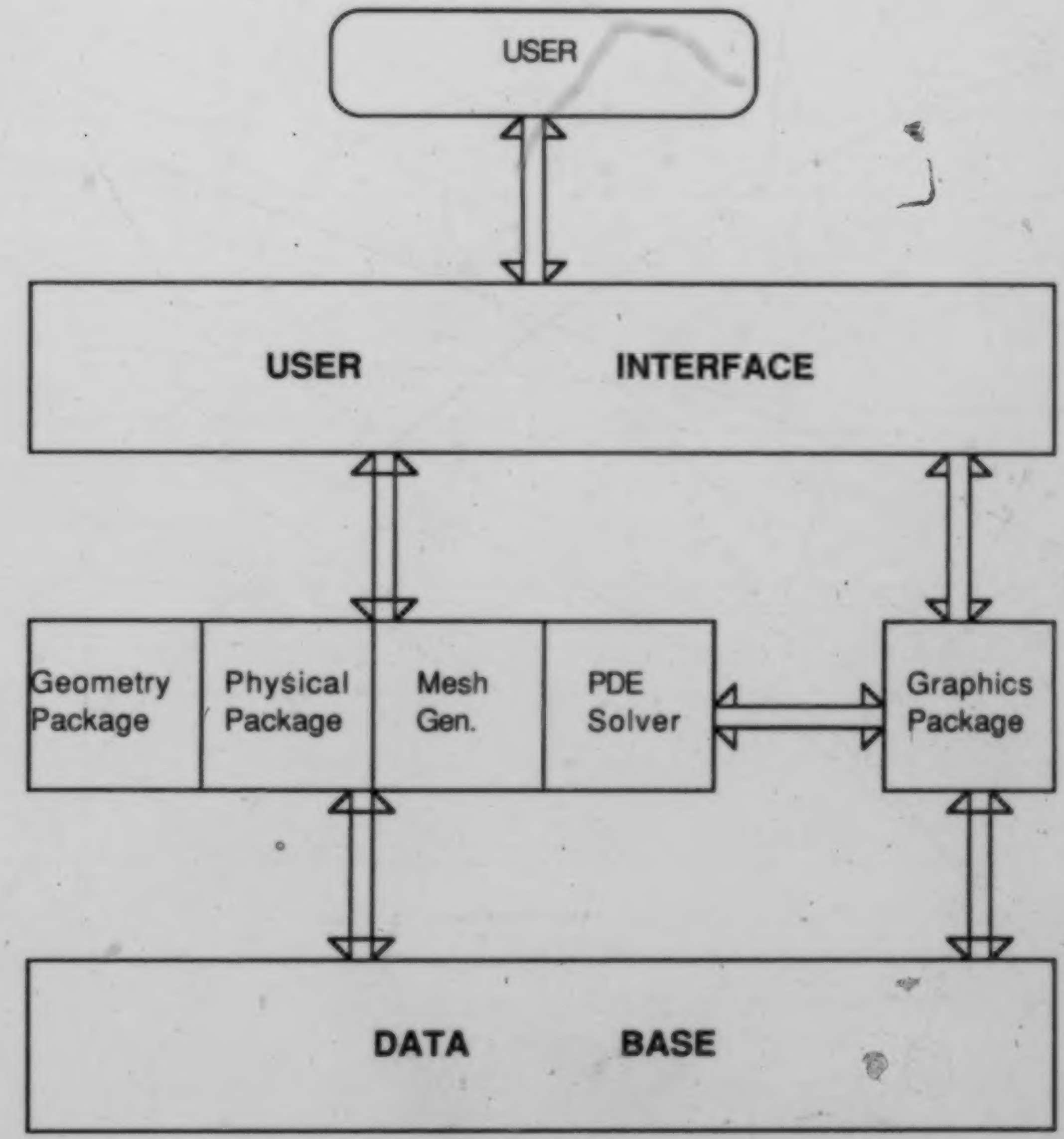

USER
USER INTERFACE
Geometry Package
Physical Package
Mesh Gen.
PDE Solver
Graphics Package
DATA BASE

Figure (4.1) : User perception of Albertina.

2. Modules layer : This layer is composed of various packages, its main function is data acquisition and processing. The source of data for each package is either another layer or a package within the layer itself. Each package residing in this layer is invoked upon a command controlled by the *user interface layer*. In Albertina the packages constituting this layer are :
    a. Geometry package : Through different modules, this package will provide a means to describe the geometric form of the device. The geometry description is preserved in a tree arrangement of well defined data structures.
    b. Physical package : This package deals with the physical description of the simulated device. It provides a library of models for each physical parameter and means of obtaining numerical values of the parameter according to a user selected model.
    c. Mesh generator: This package operates on the data provided by the two previous modules. It attempts to discretize the simulated device as described in section (2.3).
    d. PDE solver: This package is responsible for solving the non-linear algebraic equations for each mesh point. The input data are provided by the mesh generator. The output are the electrostatic distribution and the current density.
    e. Graphics package : The data resulting from any of the above packages are usually of an enormous size and are best analyzed and inspected by the user once graphically displayed. The graphics package is responsible for providing this function. It is composed of several modules each providing a different type of graphical presentation (e.g. plotting).
3. Data base layer: The user interface directs each of the second layer packages to the source of data it is to process. The data could be in this third layer which is a data pool. The pool includes data of either the current simulation session or

results of a previous session. The user could either use or inspect the contents of this layer.

Each of the above layers is extensible and independent of the other layers. Furthermore, each of the packages of the second layer are independent and separately accessible. The structure of Albertina within this context is done in order to provide the ability to modify or change any of the layers or modules, thus, achieving *modularity*.

4.2. Designer Perception of Albertina :

One of the goals we intend to achieve by restructuring Albertina is *generality*, i.e. we have defined the system structure so that it would include, but is not limited to semiconductor device simulation. This demanded a broader definition of the system. Such a definition will allow the user to define *objects* and specify sets of *actions* to be applied to them. We define this as a *Designer Perception* because such a specification is within the realm of the system designer.

The designer perception of Albertina is based on [60], where two abstract models for the *object* and *action* definitions are suggested. The incorporation of these models provide a consistent and extensible system, that would formalize user/module and module/module communication protocols. At this point, we would like to note that such an abstract approach provides a holistic view of the system. Within this view various aspects of the system are merged and are not discretely defined as in the case of a user's view of the system. The abstract approach serves as means to provide a broader definition of the problem and the solution. Thus, a wider range of application is possible. The abstract models of this approach are defined as follows :

1. Object model : The device under consideration is envisioned as a multidimensional object. One dimension is the geometric structure of the object as a whole. Another dimension is the discretized device through the mesh generator. The third dimension is physical (e.g. doping, potential and current distribution in semiconductors). The number of dimensions is set by the simulation system. These

dimensions are deduced from the equations that describe the device i.e. from the model and the solution steps. Each object dimension is a domain that could be defined and assigned a finite number of forms. For example, the geometry dimension could be defined as a circle or a polygon.

2. Action model : The manipulation of the object model requires a set of operations. Abstractly, these operations form the *action model* which is divided into two main groups, namely, definition and instantiation operations. These operations provide the following:

    a. Definition operations : Defining an object dimension means assigning a form or a model to it. For example, an object's geometry dimension could be defined as a circle i.e. a form. While the doping profile could be defined as a Gaussian distribution i.e. a model. The mechanisms and procedures responsible for such definitions constitute the definition operations.

    b. Instantiation operations : The forms/models that define an object's dimension are functions of various parameters. Relating to the above examples, the parameters of a circle are the radius and the center coordinates. For the doping profile the parameters are the dose of the implanted species and the implantation energy. In order to complete the object's description the parameters have to be instantiated with numerical values. The mechanisms and procedures responsible for this are the instantiation operations.

Figure (4.2) is an illustration of the above concepts. The figure is divided in two parts. The lower part represents the "*Abstract Level*" containing the object and action models. The object model provides the multi-dimensional description of the simulated object. These dimensions are initially neither defined nor instantiated. The operations contained in the action model provide means to the definition and instantiation of the object. The object description is stored in a data pool. The instantiation procedures of certain dimensions (e.g. discretized object) are quite complex and are separated from

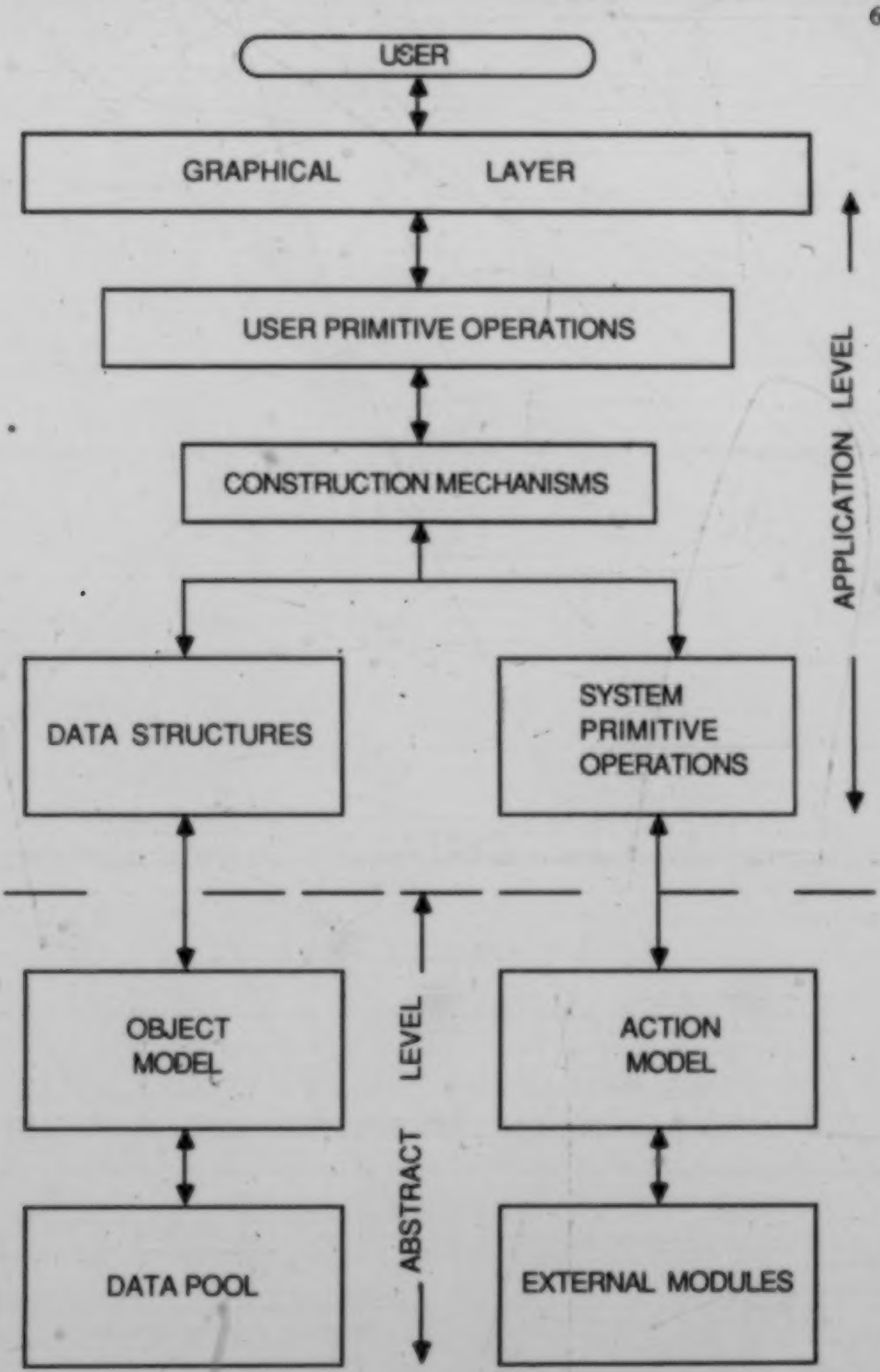

USER
GRAPHICAL LAYER
USER PRIMITIVE OPERATIONS
CONSTRUCTION MECHANISMS
DATA STRUCTURES
SYSTEM PRIMITIVE OPERATIONS
APPLICATION LEVEL
OBJECT MODEL
ACTION MODEL
ABSTRACT LEVEL
DATA POOL
EXTERNAL MODULES

Figure (4.2) : Designer perception of Albertina.

the action model. In figure (4.2) these operations are illustrated as the "External Modules".

The user interaction with the abstract level is not direct but rather through the "*Application Level*". This level is the designer's application of the user interface as shown in the upper part of the figure. The application of the action model provides a group of operations henceforth known as, *System Primitive Operations (SPO).* These operations are defined as : A group of independent operations where each is capable of manipulating only one parameter of an object's dimension. These operations are restricted to the system i.e. are inaccessible to the user. They are primitive due to their ability to handle only one parameter, hence the name System Primitive Operations. The instantiated definition of the object is stored in the "Data Structures" illustrated in figure(4.2). The description is directly manipulated by the SPOs.

Due to the primitive nature of the SPOs, their usage to manipulate the object description would be cumbersome and time consuming for a user. Therefore, the user is offered an alternative set of operations called *User Primitive Operations (UPO).* These operations are a result of algorithmic combinations of the SPOs. The creation of the UPOs require certain mechanisms to pass parameters between the SPOs of which the UPO is constructed. These mechanisms are illustrated in figure (4.2) as the medium combining the SPOs and the data structures. Finally, the user access to the UPOs is through a graphical layer. Accordingly, we can define the UPOs as a higher order construct of the System Primitive Operations. The implementation of the System Primitive Operations, data structures, construction mechanisms and User Primitive Operations are discussed in detail in chapter 5. The techniques used to implement the graphical layer are described in chapter 6.

4.3. Data Structures :

One aspect of the "Object and Action Models" operations depends on the method used to describe the simulated device. In this section we briefly introduce the data structures used in this description. The basic approach to achieve this task is to decompose the device into "Regions" where each region is abstractly described in terms of two major qualities. They are explained with reference to figure (4.3) as follows :

1. Geometric quality : This quality represents the region's geometric form and is described as a sub-tree. The nodes used in constructing this sub-tree are of the same structure (class). The attributes pertaining to each node are of a general character allowing the description of various geometric forms (e.g. circle, polygon).
2. Physical quality : This quality represents the physical aspects of the simulated object. The aspects are deduced from the model(s) used to describe the simulated object. In the case of a semiconductor device they are the physical parameters. Another class of nodes is defined to describe these parameters. This node is of a specific structure where its attributes correspond directly to the object's physical qualities. In the case of a semiconductor device each attribute represents a particular physical parameter. Thus, only one node of this class is required to describe the object's physical qualities.

In addition to these nodes we require a third class to indicate the region itself. Therefore three classes of nodes are defined as follows :

1. Region descriptor (class 0) node : As previously defined, a region is the fundamental physical and geometric entity forming the device. This node acts as a root to a tree containing both physical and geometric description of the region. A region descriptor node contains the following attributes:
    a. Name : The region's identification tag.
    b. Value : The region's characterization tag.
    c. G_link : Region's link to its geometric definition.

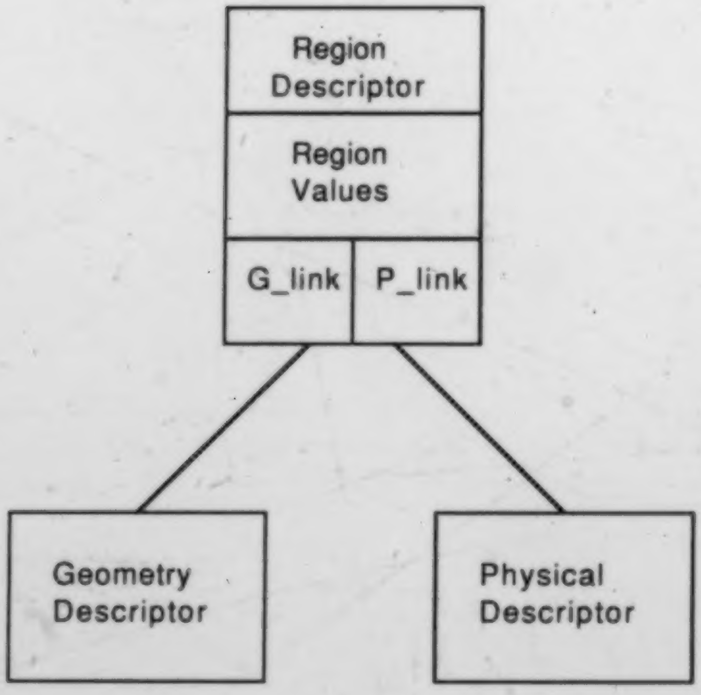

Region
Descriptor
Region
Values
G_link
P_link
Geometry
Descriptor
Physical
Descriptor

Figure (4.3): An abstract description of a "Region".

d. P_link : Region's link to its physical definition.

2. Geometry descriptor (class 1) node :

This class of nodes is used to construct a sub-tree fully describing the region's geometry. The sub-tree is appended to the region descriptor node at it's G_link. It is built within levels where any node residing in level *n* is linked to only one node in level *(n - 1)*, and to a maximum of two nodes in level *(n + 1)*. The nodes in the lower level *(n + 1)* that are linked to any *n* level node are denoted as the node's left and right children, figure (4.4). According to this configuration, we denote node *A* as the predecessor of node *F* which is node *A*'s successor. This class of nodes contains the following attributes :

a. Type : A Pointer indicating a geometric parameter related to a geometric form, e.g. *Type* = radius for a geometry form = circle.

b. Value : A Pointer indicating the value designated to the node representing the geometric parameter e.g. *Value* = 5.0 for *type* = radius.

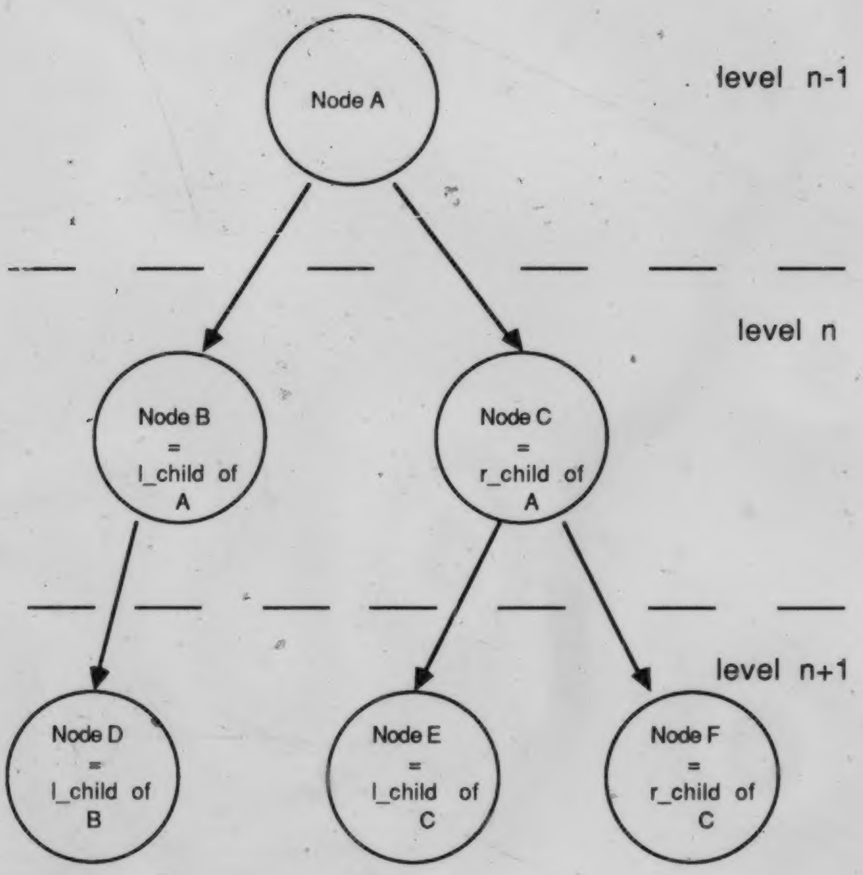
c. Left_child : Pointer to the left node in the tree which might exist depending on the geometric form under description.

d. Right_child : Analogous to the left_child a right_child might exist .

Figure (4.5) illustrates the Region and Geometry Descriptor nodes.

3. Physical descriptor node:

A node of this class is appended to each "Region descriptor" node at the p_link attribute. In a general sense the attributes of this class are a description of the physical parameters pertaining to the object under consideration; for example, it could have an attribute *material* indicating the substance of the object. Due to the fact that the relevant physical parameters of an object or a group of objects could be drastically different from each other, the data type of this class is less general than the previous two classes. It is defined with respect to the object under consideration. The attributes of this class of nodes in the case of semiconductor devices are defined in chapter 5.



level n-1
Node A
level n
Node B = l_child of A
Node C = r_child of A
level n+1
Node D = l_child of B
Node E = l_child of C
Node F = r_child of C

Figure (4.4) : Left and right child configuration for geometry descriptor nodes.

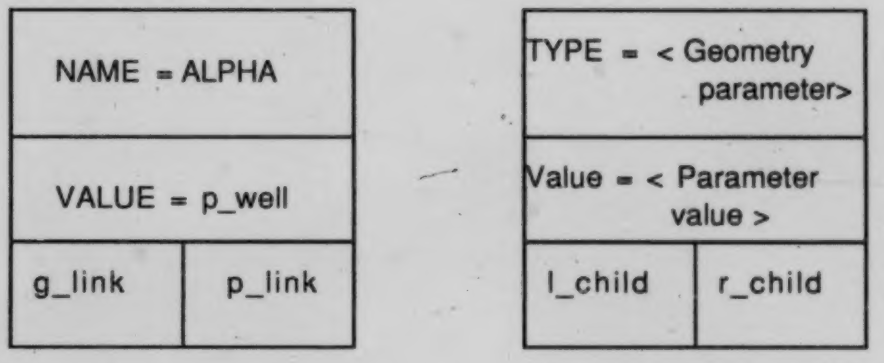

NAME = ALPHA
VALUE = p_well
g_link
p_link
TYPE = < Geometry parameter>
Value = < Parameter value >
l_child
r_child

Figure (4.5) : Graphical presentation of a Region descriptor and a Geometry descriptor.

4.4. Total Object Description :

As mentioned in section (4.3) our approach to describe the simulated device was to decompose it into regions. In the previous section we have introduced the three classes of nodes used to describe each region. The description is implemented in the form of a tree. The root of this tree is a "Region Descriptor" node linked to a sub-tree describing its geometry. The region descriptor is linked to one "Physical Descriptor" node. Thus, an assembly of these trees would provide a total description of the simulated object. During the simulation session the data stored in the description is accessed several times. Accordingly, searching mechanisms *browsers* are needed. The structure of the trees are maintained at a fairly simple and versatile level to allow fast access. Linking the region description trees is implemented with the same purpose in mind. Figure (4.6) illustrates this assembly where the device is described by three data pools, namely, Region, Geometry and Physical. The Region pool is composed of the region descriptor nodes. The nodes are arranged as a linked list, thus providing the *horizontal device description*. The "Geometry Pool" consists of separate sub-trees. All sub-trees are constructed of "Geometry Descriptor" nodes. Each tree is the geometric description of a region to which it is directly linked. The "Physical Pool", contains "Physical Descriptor" nodes of each region. The last two pools provide a more detailed description of the device as a whole and is called *vertical device description*. The pools are connected as shown in figure (4.6) indicating both description directions. The details of the *browsing* mechanisms are provided in chapter 5.

It is important to notice, that such a description has counted only for two dimensions of the device (geometric and physical). Other dimensions are not included as this work is fairly recent in comparison to the established parts of Albertina (e.g. PDE solver). Further work is required to include the other dimensions as described in section (4.2).

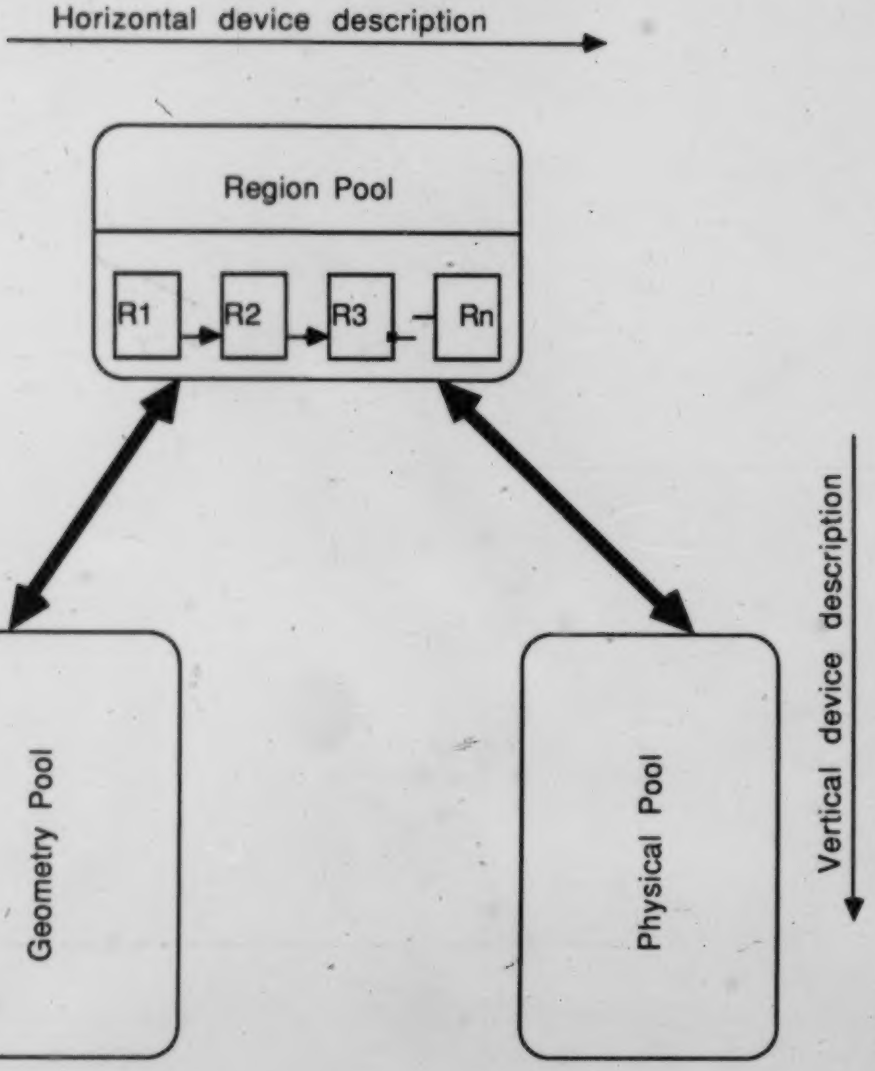

Horizontal device description
Region Pool
R1
R2
R3
Rn
Geometry Pool
Physical Pool
Vertical device description

Figure (4.6) : Total device description.

4.5. Abstract Versus Applied Perception of Albertina :

Finally, we wish to correlate the user's vision of the system as in figure (4.1) and the designer's perception of it as sketched in figure (4.2). Hence, we relate "Albertina hierarchy" and the abstract concepts. In the following points we state each Albertina package and describe its abstract counterpart as follows :

1. Geometry package : It is that part of the "User Primitive Operations" responsible for the definition and manipulation of the geometric description of the simulated object.
2. Physical package : This package is divided in three parts. The first is the "User Primitive Operations" providing means to describe the object's physical description. These mechanisms will access and instantiate the attributes of the physical descriptor node. The second, is a library of valid models per each node attribute. The third part of this package are means to calculate numerical values of each attribute according to the model assigned to it. The second and third parts are implementation details that are not included in the abstract view.
3. Mesh generator & PDE solver : In figure (4.2) they are represented as "External Modules". Within this context these modules allow the user to instantiate two dimensions of the object. The first is the discretized dimension, the second is the distribution of the electrostatic potential ϕ and the current density J. These two modules are explicitly specified in figure (4.1).
4. User interface : From figure (4.1) the user perceives the interface as a separate package. In this case, the interface provides methods for the user to communicate with the "Modules Layer", and means to access the "Data Base". From this point of view, the user expects the interface to provide various communication protocols and monitor the interaction between the various modules. From the designer view point this distinction is not that simple. This is due to the interactive and graphical techniques used to implement Albertina's user interface as discussed

previously in section (2.6). Using such techniques results in a merger between the package and the interface. We have experienced this fact during the development of this work. From the designer perspective the interface is a number of invokable windows. Each window offers a user the ability to *see* and *control* a certain process. These windows are connected in a hierarchy, which could be viewed as tree. The root of such a tree is the graphical layer shown in figure (4.2). In the graphical layer, each operation is presented by a panel. If a panel is selected by the user, a new window is created and the panel's relevant operation is invoked within the window. Therefore, in figure (4.2) the user interface is represented by the graphical layer, through which the "User Primitive Operations" are presented in the form of panels. Due to the abstract description provided in figure (4.2), windows and UPO invoking mechanisms are not represented as they are implementation details.

5. Graphics package : In figure (4.1) it is the module(s) responsible for displaying one facet of the object or another. Since these types of modules are "application specific", i.e. dependent on the hardware and software used to implement the system, they are implicitly considered in an abstract model such as figure (4.2). Further details of this package are provided in chapter 6 which deals with system implementation.

5. Application of the User Interface

In this chapter we discuss the implementation of the abstract concepts upon which Albertina was restructured, namely the implementation of the "Action" and "Object" models. The nodes used to describe the simulated device are further explained. We then proceed to give a detailed definition and explanation of the different operations provided by the system and the algorithms used in the construction of the operations provided to the user.

5.1. Object Model Application :

The application of the "Object Model" concept to semiconductor device simulation requires definition and specification of :

1. Sub-tree structures describing the geometric forms implemented by Albertina, in a two dimensional Cartesian plane.
2. A precise and comprehensive definition of the data type representing the "Physical Descriptor" node.

5.1.1. Geometry Description :

In the implementation of Albertina, we have considered a group of geometric forms. Each form is represented as a sub-tree of class 1 nodes. The general attributes of this class are identified in section (4.3). Accordingly, the attribute *type* of each node within the sub-tree has to be defined in order to specify a certain geometric form. The definition of a node *type* corresponds to the parameters of the geometric form, as follows :

1. Circle : Defined by (x_o, y_o) coordinates of the center and a radius r. Therefore three types of class 1 nodes are defined as *xcr, ycr* and *rd,* to express x_o, y_o and the radius respectively.

2. Rectangle : Defined by (x_o, y_o) and (x_1, y_1) the coordinates of the right lower (bottom) and the left upper (top) corners of the rectangle. The corresponding types of class 1 nodes are *xbm, ybm* and *xtp, ytp*.
3. Polygon : Defined as a series of points (x_n, y_n), where $n = 0, 1, 2, 3.... i$, as i is the total number of the polygon's vertices. Each point is represented by nodes of type *xpol* and *ypol*.
4. Ellipse : Defined by coordinates of the ellipse center *(xc, yc)* the minimum and the maximum axes *(minx, miny)*. The four ellipse parameters are defined as node types of the sub-tree describing an ellipse.
5. Pocket : This geometric form represents a diffusion region in a semiconductor device. It is described by an elliptic arc. The types *xst* , *yst* define the left most point on the elliptic arc, and *xcr, ycr* define the center of the arc. The depth of the pocket is defined at 0.8 µ*m* as per the technology standard.
6. Curve : A curvature in Albertina is represented in a polynomial form :

$$y = f(x) = a_0 x^{\sigma_1} + a_1 x^{\sigma_2} + a_2 x^{\sigma_3} + ... , \tag{5.1}$$

where a_0, a_1, a_2 are real numbers of arbitrary values, and the exponents $\sigma_1, \sigma_2, \sigma_3....$ are integer fraction numbers. The curvature is assumed continuous between x_i, x_j. A tree describing such a polynomial will have nodes of class 1 of the type *coef* indicating the coefficients a_o, a_1...etc., and *pow* representing the power of x $\sigma_1, \sigma_2, \sigma_3....$ etc. Two more types to express the lower and upper limits x_i, x_j, are required and are defined as *xlow, xup*.

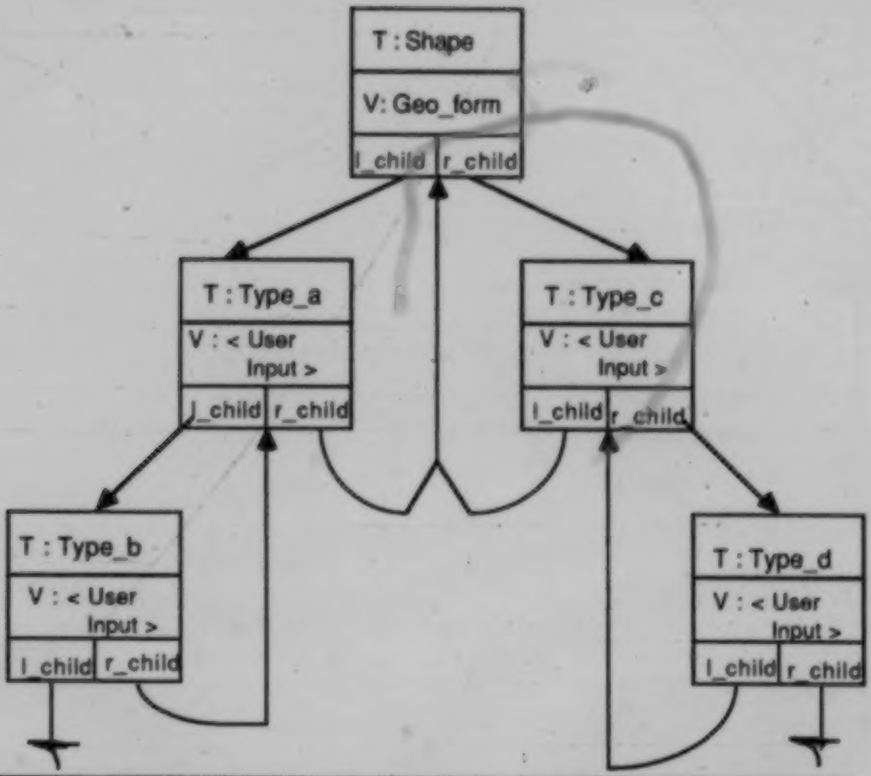
We finally need one more *type* of node, that will act as the root for the sub-tree geometric description of the region, and will be responsible to identify the geometric form represented by the sub-tree. A node of this type would be the only one of this class linked to the "Region Descriptor". This node will be of type *shape* and the *value* attribute of this node will be any of the above geometric forms (e.g. circle).

Having defined all of the above types, we move on to define the structures of the sub-trees describing each geometric form. To minimize the complexity of the mechanisms manipulating the sub-trees, we maintain the following two rules :

1. Uniformity : In order to universally apply the manipulating mechanisms to all geometry sub-trees, they have to be built in a similar and uniform structure.
2. Simplicity : The complexity of searching for a particular node within a sub-tree is directly proportional to the complexity of the tree structure, therefore, to reduce the search complexity we selected a simple tree structure that would suit all the geometric forms.

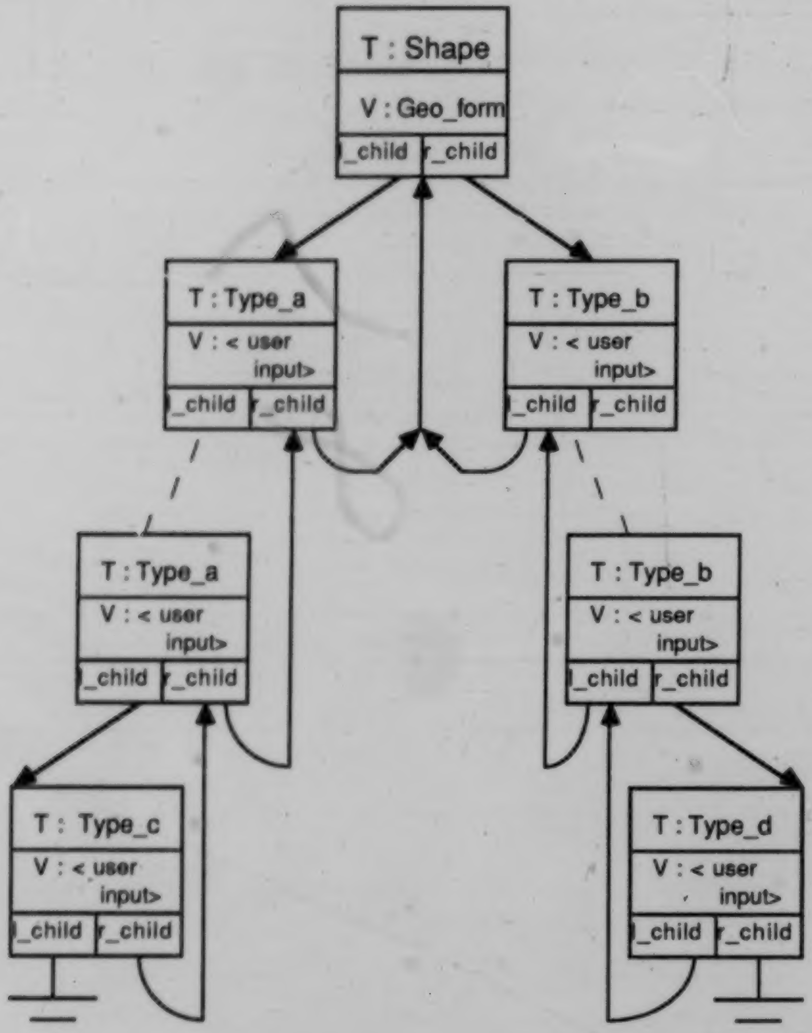
Using these rules a geometry sub-tree is constructed of class 1 nodes. The construction is according to the configuration discussed in section (4.3) and figure (4.5). The general structure of the geometry sub-trees is as follows :

A node of this class, denoted as the root, is connected to a region descriptor node. The sub-tree connected to the root is a description of the region's geometric form. The root is of type *shape* and *value = geometric form*, i.e. if the region is a circle *value = circle*. With respect to the region descriptor, the root node resides in level 1. Thus, the root's children node reside in levels *2,3,4, ... Z*. In this case the subtree is of depth *Z*. The root has two children linked at *r_child, l_child*. Nodes residing in level *x* are children of those in level *(x-1)* and so on until level *Z*. All nodes have one child except the root which has two. A child node is linked to a parent at the parent's *x_link*, where *x_link* is the same as the root's link to the node's predecessor residing in level 2. For example, if node *B* in level *x* is a successor of a node *A* and has a child *C* residing in level *x*+1, where *A* resides in level 2 and is the root's *r_child*. According to the sub-tree structure node *C* will also be node *B*'s *r_child*. Hence, each node will have a free link. The unused link is *threaded* back to the node's parent. The threads are used to perform traversals, insertions and deletions without using either a stack or recursion. According to the above, geometric sub-trees are illustrated in figures (5.1) and (5.2). The first figure represents the first group of sub-trees, which is of a defined depth.


T : Shape
V: Geo_form
l_child
r_child
T : Type_a
V : < User Input >
l_child
r_child
T : Type_c
V : < User Input >
l_child
r_child
T : Type_b
V : < User Input >
l_child
r_child
T : Type_d
V : < User Input >
l_child
r_child

Geo_form	Type _a	Type_b	Type_c	Type_d
Circle	ycr	N/A	xcr	rd
Rectangle	xbm	xtp	ybm	ytp
Ellipse	yc	miny	xc	majx
Pocket	xst	yst	xcr	ycr

Figure (5.1) : Circle-Rectangle-Ellipse-Pocket subtrees.


T : Shape
V : Geo_form
l_child
r_child
T : Type_a
V : < user input>
l_child
r_child
T : Type_b
V : < user input>
l_child
r_child
T : Type_a
V : < user input>
l_child
r_child
T : Type_b
V : < user input>
l_child
r_child
T : Type_c
V : < user input>
l_child
r_child
T : Type_d
V : < user input>
l_child
r_child

Geo_Form	Type_a	Type-b	Type-c	Type_d
Polygon	ypol	xpol	--	--
Curve	coef	pow	xlow	xup

Figure (5.2) : Curve-Polygon subtree.

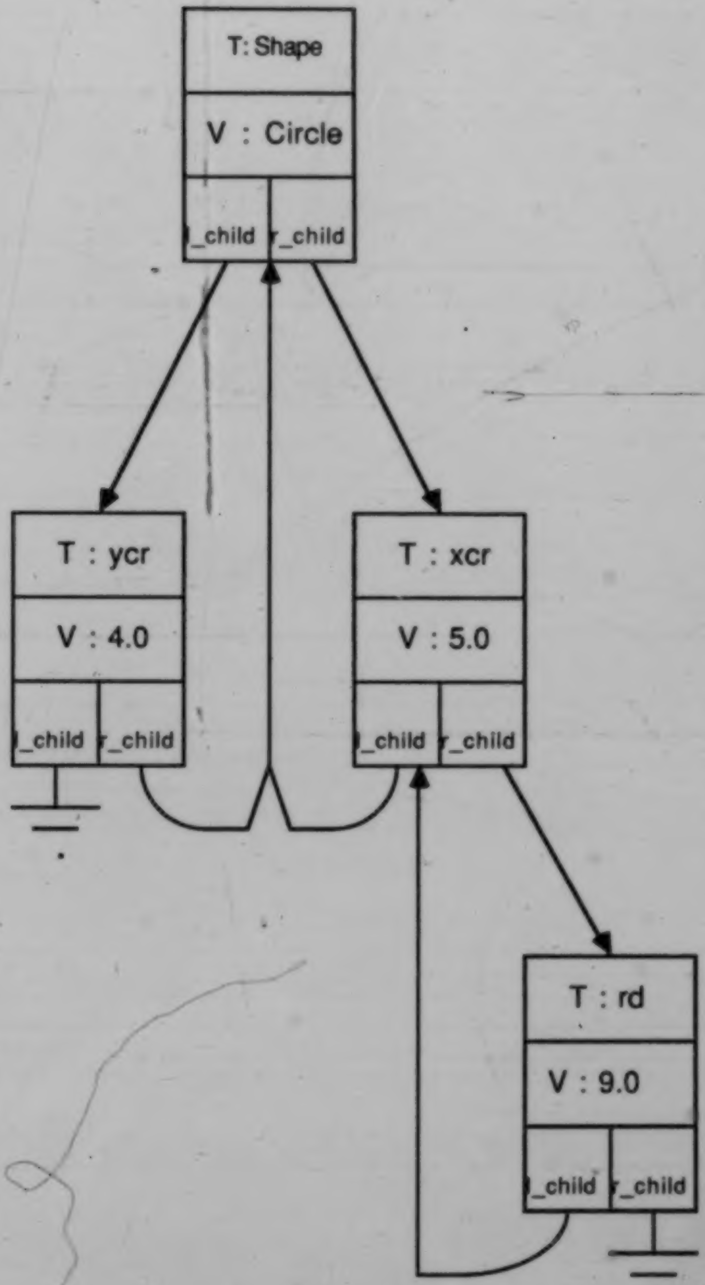

T: Shape
V : Circle
l_child
r_child
T : ycr
V : 4.0
l_child
r_child
T : xcr
V : 5.0
l_child
r_child
T : rd
V : 9.0
l_child
r_child

Figure (5.3) : Tree presentation of a circle
$(x-5.0)^2+(y-4.0)^2=(9.0)^2$

They are used to describe a circle, rectangle, ellipse and pocket. Node types depend on the geometric form represented by the sub-tree. The geometric form is the value of the root and is denoted by *Geo_form*. Accordingly, the types are defined as per the table in figure (5.1). The second figure (5.2) represents curves and polygons. The structure of the sub-tree represented by this group is the same as the previous group. In figure (5.2) a table listing the type of each node for a certain *Geo_form*. is provided. The difference between the two groups is the sub-tree depth. For the second group the sub-tree depth is undefined. The reason is that neither the number of polygon vertices nor the polynomial terms are pre-defined by the system. The sub-tree depth in this case is user defined. Further discussion of the construction of geometric sub-trees is provided in section (5.5). A detailed illustration of a geometric sub-tree representing a circle is given in figure (5.3).

5.1.2. Physical Description :

As previously discussed, the data type of the "Physical Descriptor" node has to contain a particular set of physical information pertaining to the simulated object.

In the case of semiconductor devices, and as discussed in detail in chapter 3, the "Physical Descriptor "node is a direct presentation of the physical parameters. Hence it has the following attributes:

a. Bandgap narrowing : Denoting a bandgap narrowing model.

b. Temperature : Denoting the temperature of the region in Kelvin scale.

c. Dose : Denoting the implanted dose of impurities within the region.

d. Carrier mobility : Denoting a user selected carrier mobility model.

e. Doping profile : An indicator to a user selected impurities distribution function.

f. Generation and Recombination : A pointer to a model or sum of models (see section 5.4) representing the generation and recombination phenomena.

g. Dopant : An indicator to the species of the impurity implanted in the region.

h. Boundary : A pointer indicating the type of boundary condition the user has selected for the region to use during the execution of the partial differential equation solver.

A definition of the three classes of nodes is listed in appendix I.

5.2. Action Model Mechanisms :

The mechanisms provided in Albertina to build the higher order constructs were defined in section (4.2). These mechanisms are classified in terms of two categories :

1. Node referencing mechanism : This mechanism provides means through which nodes could be created or located; the mechanism references the node through pointers that are returned to the caller.
2. Tree browsing mechanism: As the device description is dynamically implemented in a tree structure, this mechanism will provide its caller with either the total device description or the description of only one region.

The following sub-sections discuss these mechanisms.

5.2.1. Node Referencing Mechanism (NRM) :

In the process of building and manipulating a tree describing a device, we require a mechanism allowing us to reference each node of the tree. A most efficient method of accomplishing this task is through the use of **pointers** defined as [63] : A variable that gives the location of another variable. Hence, using pointers we create the Node Referencing Mechanism which is the fundamental mechanism used in Albertina to create and locate any class of nodes. This mechanism is defined by two functions each returning a pointer to the referenced node. Those functions are :

1. Allocate node : Using *dynamic memory allocation* techniques, for requesting additional memory inherent in most programming languages, we implement the *allocate* function which allows the creation of a new node of any class. *allocate* will

return a **node_pointer** to its caller, and has the following synopsis :

node_pointer = allocate (node_class, node_type, node_name)

Where the **node_class** is an integer (0, 1, 2) indicating the class of the *allocated* node. If the allocated node is a "Region Descriptor", the argument **node_name** is used to instantiate the node's *name* attribute. A Geometry Descriptor's *type* attribute is instantiated by the argument **node_type.**

2. Locate node : This mechanism provides its caller with a pointer to a particular node of any of the three classes. As nodes of class 1 and 2 are appended to their relevant "Region Descriptor" node, the *name* attribute identifying the region has to be provided in order to find the root of the tree describing the region. A second attribute to be defined is *node_class*. If *node_class = 1*, *locate* will return a pointer to the "Region Descriptor" node. If *node_class = 2* a pointer to the region's "Physical Descriptor" node is returned. In case *node_class = 1* the search will proceed through the Region descriptor's G_link into the geometry sub-tree; hence the caller has to define the node's *type* as a third attribute.

   Locate is of the following synopsis :

   node_pointer = locate (region_name, node_class, node _type)

For example, if we need to locate the physical descriptor node of a region named ζ, *locate* will be called as :

pointer1 = locate (ζ, 2, NULL)

/* name = ζ, class = 2, type = NULL as it is not applicable to class 2 nodes */

If ζ has a geometry description of a "Circle", we reference the "radius" node as follows :

pointer2 = locate (ζ, 1, rd) /* where rd is the **node_type** */

A NULL pointer is returned when either the region, or the node pertaining to it do not exist.

5.2.2. Tree Browsing Mechanism :

On invoking certain operations (e.g. *Save*) the system will require a mechanism that would indiscriminately propagate either through a region or the description of the simulated device as a whole. This mechanism would provide pointers for all encountered nodes to a certain destination. Such is the task of the Tree Browsing Mechanism TBM. It basically could be seen as an iteration of the *locate* SPO but does not search for a particular node.

In the case of *browsing* through the whole device we need to propagate in the following directions (see figure 4.3) :

1. Horizontal Device Description, i.e through all "Region Descriptors".
2. Vertical Device Description i.e through the sub-tree geometric and physical description of each region.

TBM is accordingly divided into two mechanisms:

- a. Horizontal Propagation Procedure : Allowing a *region_pointer* to travel horizontally through the "Region Descriptor" linked list.
- b. Vertical Propagation Procedure : Starting at a Region Descriptor node as a root. This procedure will pass a *region_pointer* to the *destination*. followed by a pointer to the region's physical descriptor node. The procedure will then travel through the geometry sub-tree. The travelling starts at the g_link node, first through each left_child, then through each right_child, until a "NULL" node is encountered.

As these mechanisms are implemented in the "C" programming language we use this language's capability of defining a pointer to a function to implement this mechanism. Keeping in mind the structure of the geometry sub-trees and denoting the destination procedure as *my_function*. A pointer to each encountered node will be returned to *my_function*. TBM for the whole device is an implementation of the following algorithm:

```
var destination : @my_function
/* Assign the variable destination as a pointer to "my_function" */
procedure browse ( destination )
     var rg_ptr : region_pointer;
begin
     rg_ptr = get_first_region();
     /* Initialize to the first region Descriptor */
     while not rg_ptr = NULL
          /* Loop until the end of the Region Descriptor linked list */
     do destination ( rg_ptr);
          /* Through the destination_pointer pass the region_pointer to my_function */
          v_browse(destination, rg_ptr);
          /* Pass both pointers to the vertical browsing mechanism */
          rg_ptr = rg_ptr. R_Next;
          /* Move to the Next Region Descriptor node */
          end;
end
procedure v_browse ( destination, rg_ptr)
     var p_ptr : ppar_pointer ; g_ptr : geo_pointer;
     begin
          p_ptr = rg_ptr.plink;
          /* Originate the pointer to the region's Physical Descriptor node */
          destination ( p_ptr);
/* Through the destination_pointer pass the Physical_pointer to my_function */
```

```
        g_ptr = rg_tr.g_link;
        /* Originate the pointer to the root of the geometry sub-tree */
    while not g_ptr = NULL;
        /* Until a NULL node travel through the left hand side of the
        geometry sub_tree */
        do
            destination ( g_ptr);
            /* Pass every pointer of the sub_tree to my_function */
            g_ptr = g_ptr.l_child;
            /* Move to the next left child */
    end
        g_ptr = rg_tr.g_link;
        /* Re-initialize the root of the sub_tree , then travel right */
    while not g_ptr = NULL;
        do
            destination ( g_ptr);
            g_ptr = g_ptr.r_child;
    end
end
```

Part of this algorithm is utilized to browse through one region only if it is required either by the user or the system. In such a case the following algorithm is applied :

```
function region_browse (region_name, destination)
var rg_pointer : region_pointer;
begin
```

```
    rg_pointer = locate (region_name, 1, NULL);
    v_browse ( destination, rg_pointer);
end
```

5.3. System Primitive Operations (SPO) :

As previously defined the "System Primitive Operations", are the fundamental operations that allow the physical and geometric description of the simulated device. These operations manipulate nodes only, i.e. they are not concerned with the subtree attached to the node. Node manipulation is implemented through pointers returned from the "Node Referencing Mechanism" functions, *allocate* and *locate*. Nodes of the previously defined classes are subject to the following "System Primitive Operations" :

1. Alter node : During the course of a simulation session the attributes of every created node is instantiated with a certain value. Instantiation occurs either upon the creation of the node or as a result of a user instruction. We therefore require a universal procedure that will *alter* the value of an attribute pertaining to a particular node. This SPO is implemented using the **node_pointer** returned from either NRM or TBM. It has the following synopsis :

   alter(node_class, node_pointer, attribute, new_value)

   node_class : integer; /*Denoting the node's class */

   attribute : @char; /* Pointer to the attribute to be altered */

   new_value : @char; /* Pointer to the attribute's new_value */

2. Transcribe node : This primitive operation is responsible for the duplication of an entire node including the values of its attributes to another node. The transcribed node denoted as **from_node,** is copied to **to_node,** therefore, both nodes must be of the same class. The responsibility of locating both nodes and creating the **to_node** in case it does not exist is beyond the function of this primitive operation and is implemented at a higher level of sophistication, namely , the "User

Primitive Operations". Accordingly, the synopsis of this operation is as follows :

transcribe (to_node , from_node)

3. Erase node : As the simulation session proceeds the user might need to change the geometric description of a certain region or even remove it. Therefore, for the sake of efficient memory use, we need a primitive operation at the node level to erase each node, which basically returns to the system the memory space used by the node, i.e. it is the inverse of the *allocate* SPO. It has the following synopsis :

   erase (node_pointer)

4. Preserve / Retrieve node : This final primitive operation allows the system to preserve or retrieve a particular node and its attributes. It does encompass both operations, as they could be seen as one operation with the only difference being the direction of data flow between the node and the storage location. This operation is of the following form :

   Pres_ieve (direction, node_pointer, storage_pointer)

In this situation the argument **direction** represents the direction of the data flow and has one value for *Preserve* and another for *Retrieve*. One more aspect of this idea which has to be considered is data format transformation between the storage format and the tree structures of the nodes. The complexity of *Pres_ieve* appreciably increases upon the incorporation of the format transformation aspects. As a result, it could no longer be classified as a "System Primitive Operation". Based on this idea, analogous "User Primitive Operations" are built to provide this functionality.

5.4. Application of the Action Model :

As discussed above, the combination of "System Primitive Operations" with the "Node Referencing" and the "Tree Browsing" mechanism, within well defined algorithms creates a group of operations labelled "User Primitive Operations" UPO. In the following sub-sections we discuss each UPO and the algorithms used in its implementation.

5.4.1. User Primitive Operations (UPO) :

Albertina's user interface provides a group of UPOs at a fairly low degree of complexity. This is done with the assumption that the user is at a high level of expertise and therefore requires a high degree of freedom. UPOs are classified into two groups according to the object or sub-object (region vs. device) they manipulate. These groups are :

1. Region operations : In this group the operations are directly concerned with region manipulation. The UPOs accomplishing these tasks are:
    a. Create : Responsible for creating a region. If the user provides a name for the newly created region, it will be used; otherwise the system will provide a name and notify the user. *Create* is implemented according to the following algorithm :

```
const region_class = 0; gpar_class = 1; ppar_class = 2;
/* Define the node classes */
procedure create_region(given_name)
var name : char ;
begin
        if given_name = NULL ; /* No user defined name */
        then name = get_name();
        /*Get new name from the system */
        region_pointer = allocate ( region_class, NULL, name);
        /* Create a Region Descriptor */
        gpar_pointer = allocate ( gpar_class, "shape", NULL);
        /* Create a Root node for the geometry sub tree */
        ppar_pointer = allocate ( ppar_class, NULL, NULL);
        /* Create a Physical Descriptor */
        region_pointer. g_link = geo_pointer;
        /* Link the Region Descriptor and the Geometry root */
        region_pointer. p_link = ppar_pointer;
        /* Link the Region and the Physical Descriptors */
        print_message_to_user("Region created");
        /*Notify user of success and name of new region */
end
```

This algorithm is explained in the following steps:

i. Check if user provided a name, if not then get a new name from the system.

ii. Allocate a region descriptor node.

iii. Allocate a geometry descriptor node of type *shape* and link it to the region node at *g_link*.

iv. Allocate a physical descriptor node, and link it to the region node at *p_link*.

b. Name : After *creating* a region the user might want to change its name i.e. re-instantiate the region descriptor's *name* attribute. Therefore, the user has to specify two arguments, the region's **old_name** and **new_name** in order to implement the following algorithm :

```
procedure (old_name : @char, new_name : @char)
const region_class = 1 ; att = 'Name';
var rg_ptr : region_pointer;
begin
```

```
        rg_ptr = locate (old_name, region_class, NULL);
        alter( region_class, rg_ptr, att, new_name);
end
```

This algorithm is explained in the following steps :

i. *locate* the region using the old name.

ii. Pass the pointer returned from the previous step to the "alter" SPO, with the new name.

c. Type : This operation is used to instantiate the "Region Descriptor" attribute *type*. It is implemented using the same algorithm as in *name* SPO after specifying the attribute as :

```
        var att = 'Type';
```

d. Delete : Using TBM and directing the *destination_pointer* to the *erase* SPO, a whole region is deleted by applying the following algorithm :

```
var destination : @erase();
procedure delete (destination: function_pointer, region_name :@char)
var rg_ptr : region_pointer;
begin
        rg_ptr = locate (region_name, 0, NULL);
        v_browse ( destination, rg_ptr);
end
```

The algorithm is further explained as :

i. *locate* the region using the user provided *region_name*.

ii. Pass the returned pointer to the *v_browse* mechanism which will use the pointer to propagate through the region. This mechanism will provide *erase* with a pointer to each encountered node.

iii. *erase* will then free the memory space occupied by each node linked to the region.

e. Copy : Using the *transcribe* SPO concept, this operation will duplicate a user named region denoted as **from_region** under another user specified name referred to as **to_region.** The sequence through which this operation is implemented is as follows :

```
procedure ( to_region: @char; from_region : @char )
var destination : function_pointer;
    to_ptr : region_pointer;
    from_ptr : region_pointer;
    root : geo_pointer;
begin
    from_ptr = locate (from_region, 0, NULL );
    if from_ptr = NULL ;     /*from_region does not exist */
    then print_error_message();
    else to_ptr = locate (to_region, 0, NULL );
    if
    to_ptr = NULL;           /*If the to_region does not exist */
    then to_ptr = allocate (0, 0, from_region); /* Create it */
    else root = locate (from_region , 1, "shape");
        destination = @erase();
        v_browse (destination, root);
        /* Delete to_region old geometric sub_tree */
    end
```

```
        destination = @transcribe();
        v_browse ( destination, from_ptr );
end
```

We explain this algorithm with the following steps :

i. *locate* the **from_region** and return an error message if it does not exist.

ii. *locate* the **to_region,** if it does not exist, *create* a new region and *name* it according to the user given name.

iii. For a region that already exists, delete its geometric description subtree.

iv. Finally use the *v_browse* mechanism to propagate through the region, through which pointers to all encountered nodes will be passed to *transcribe*.

f. Shape : This operation is responsible for designating a certain geometric shape to a region. The shape is selected by the user. This is done by appending the appropriate geometry sub-tree to the region descriptor at it's g_link. This operations is implemented in the following sequence :

```
shape ( region_name: @char; geo_form : @char )
var rg_ptr: region_pointer;
        geo_root: geo_pointer;
begin
        rg_ptr = locate (region_name, 0, NULL );
        /*Find the named region*/
        geo_root = rg_ptr.g_link;

        /*Move to root of geometry sub_tree */
        alter ( 1, geo_ptr, "value", geo_form );
        /* Instantiate the root value attribute value = geo_form */
        case geo_form of
            known_geo_form: tree_building_mech (geo_form, root_ptr);
/* Depending on geo_form, pass the root pointer to the appropriate tree
building mechanism */
end
```

The following steps are the explanation of the above algorithm :

i. *locate* the region using the user provided name.

ii. Using the pointer returned from the *locate* operation, originate a new pointer (g_pointer) to the first geometry descriptor node of type *shape*.

iii. Instantiate the *shape* node using *alter* with g_pointer as its argument, so that the node value would be the geometric shape.

iv. According to the geometric shape parameter, pass g_pointer to the appropriate tree building mechanism, where the tree describing the geometry will be created and appended to the *shape* node.

g. Ishape : This operation is the means by which the user will instantiate the values of each node of the sub-tree describing the geometry of a region. In this case, some input-output aspects should be considered. These aspects will be discussed in a later chapter. *Ishape* is implemented by applying the following algorithm :

```
procedure ishape (region_name )
var rg_ptr : region_pointer;
    geo_ptr : geo_pointer;
    destination : function_pointer;
begin
    rg_ptr = locate (region_name, 0, NULL );
```

```
        /*Find the named region*/
        geo_ptr = rg_ptr.g_link;
        destination = @in_out();
        v_browse (destination, geo_ptr);
    end
    procedure in_out ( ptr : geo_pointer )
    var buffer : @char;
    begin
        display_to_user(geo_pointer.type);
        buffer = get_user_input();
        alter (1, geo_pointer, "value", buffer);
    end
```

Further explanation of the algorithm is :

i. *Locate* The region using the name provided by the user.

ii. Move to the root of the geometry sub-tree node of type *shape*, read the node value and identify the geometric form.

iii. Direct the *destination_pointer* to the input-output routine, and pass it to *v_browse*.

The I/O routine will process the node pointer as follows :

a. Display the node type to the user .

b. Prompt the user for a value.

c. Buffer user input.

d. Upon input termination, pass the node_pointer and the buffer as arguments to the *alter* SPO.

h. Ppar : In an analogy to *Ishape* which is responsible for instantiating the geometric form of a region, ***Ppar*** is the operation responsible for instantiating the physical parameters pertaining to a particular region. Although the idea is essentially the same, the instantiation mechanism is different due the nature of the values of the physical parameters. As previously explained, the value of each attribute in a physical descriptor node is a pointer to a particular model. The model designated to an attribute is selected by the user from the "Library of Physical Parameters". This library is built in the system and is available to the user through a mechanism of menus which are built in a hierarchy. The first is the "Attribute Menu" which allows the user to select a Physical Descriptor attribute. Once this attribute is selected the user accesses the "Model Menu" allowing him to assign a model to the attribute. *Ppar* will operate in the following sequence :

```
procedure ppar( region_name)
var p_ptr = ppar_pointer;
    rg_ptr = region_pointer;
    att = @char;
    model = @char;
begin
    rg_ptr = locate (region_name, 0, NULL );
    /*Find the named region*/
    p_ptr = rg_ptr.p_link;
    att = get_attribute();
    model = get_model();
    alter (2, p_ptr, att, model);
end
```

The following steps explain this algorithm :

i. *locate* the user named region .

ii. Through the p_link attribute of the located region descriptor, originate a pointer **p_pointer** to the physical descriptor node.

iii. Upon user invokement of the physical parameter "attribute menu", buffer the selected attribute **att** .

iv. Upon user invokement of the physical parameter "model menu", buffer the selected model **model** .

v. Using **p_pointer** *alter* the attribute, using the argument **model** as the new value, and **att** for attribute identification.

Detailed discussion of the "attribute menu" and "model menu" is given in chapter 6, based on the discussion of models in chapter 3.

i. *Save*: As discussed in the *transcribe* SPO, a *format_conversion* procedure is required to change the data from the tree structure to the storage format. This procedure requires one argument which is a pointer to the *storage_destination*. In this case it is a pointer a file. The description of one region is *Saved* as follows :

```
var storage_destination : storage_pointer
/* A pointer to the storage location */
procedure save (region_name : @char,
          storage_destination : storage_pointer)
var rg_ptr : region_pointer ;
          destination : @function_pointer ;
begin
          rg_ptr = locate (region_name, 0, NULL);
          /* Use the locate SPO to find the region*/
```

```
        destination = @format_converter (storage_destination);
        /* Direct the file pointer to the format conversion procedure,
        set the destination before vertical browsing*/
        v_browse (destination, rg_ptr);
    end
```

We explain this algorithm as follows :

i. The specified region is *located*.

ii. The second step is to pass the file pointer to the format_converter procedure. This procedure is meanwhile assigned as the destination of the vertical browsing mechanism.

iii. The vertical browser is called, thus, the destination pointer will propagate through the region's description. The format converter will then receive a pointer to each node. The node attributes are then written to the storage location.

In case the whole device requires saving, this procedure is iterated within an *h_browse* loop.

j. List : As the user would like to inspect the data or structure of a particular region, this UPO will display the region on the standard output. Therefore, the same algorithm used to implement *save* UPO will accomplish this task after, redirecting the *storage_destination* pointer to the standard output (i.e. the computer terminal).

k. Load : The *format converter* used by the *save* UPO will transform the device description from a tree structure to a sequential file, where each record in the file is composed of several fields in the following format :

Field 1	Field 2	Field 3	Field 4
Node_id	Node att1	Node att2	Node att3

Table (5.1) : Format of "Save".

To explain the format, a sample file is given in the following table :

Node_id	Node att1	Node att2	Node att3
00	device	lala	CMOS
10	α	substrate	
20	shape	Rectangle	
21	xbm	520	
21	xtp	20	
22	ybm	50	
22	ytp	200	
31	Slot'_Degraaf		
32	300		
33	1E15		
34	Arora		
35	1D_Gaussian		
36	Auger		
37	P		
38	Neumann		
10	β	well	
20	shape	Pocket	
21	xst	470	
21	xcr	485	
22	yst	200	
22	ycr	200	
31	Slot'_Degraaf		
32	300		
33	1E19		
34	Selberherr		
35	1D_Gaussian		
36	SUM		
37	B		
38	Dirichlet		

Table (5.2) : Sample of saved device.

The above file contains data describing a CMOS device named "lala" which is built of two regions. The first region is a substrate named α, which is of a rectangular shape. In the file the geometric and physical description of the region follows. The second region is a well named β, of a "Pocket" shape and is followed by a listing of its geometry and physical definition.

In this file the node_id field is composed of two digits x and y. The first digit x, identities the **node_class** i.e x = 1 for a region descriptor node, x = 2 for a geometry descriptor node and x = 3 for a physical descriptor node. The interpretation of the second digit y is dependent on the first as follows :

i. With x = 1 :

- If y = 0 then the node is the root of the geometry subtree.
- If y = 1 indicates that this node is a *left_child* of the previous one.
- If y = 2 then the node is a *right_child.*

ii. With x = 3 the node is a *physical descriptor* and the second digit y is an attribute indicator, as per table (5.3) :

value of y	Attribute
1	Bandgap Narrowing
2	Temperature
3	Implanted Dose
4	Mobility
5	Doping Profile
6	Generation & Recombination
7	Dopant Species
8	Boundary Condition

Table (5.3) : Identification of class 3 node attributes.

Since we have previously determined that a *load* UPO is the inverse of a *save* UPO, *load* will use the above file to rebuild the tree structure describing the device. *Load* will use x and y of the node_id to reconstruct each region of the device into a tree. Moreover, node_id = 00 indicates a device cluster record, see figure (5.5). When x = 1 the node is a geometry descriptor thus y is ignored.

The operation of *load* is done with the following algorithm:

```
procedure load_dev (file_pointer)
    var class_id :integer;
        p_att : integer;
        att1 : @char;
        att2 : @char;
        att3 : @char;
        region_name : @char;
        region_type : @char;
        rg_ptr : region_pointer;
        g_ptr : geo_pointer;
        p_ptr : ppar_pointer;
begin
while not eof
    do
    get_all_fields (file_pointer, class_id, att1, att2, att3);
    if class_id = 0 ;
        then create_device_cluster( att2, att3);
    else
        if class_id = 10;
            then region_name = att1;
                region_type = att2;
                rg_ptr = creat_region ( region_name);
                alter (1, rg_ptr, region_type, "Type");
        else if class_id = 20;
            /* if second digit =  0, this is the root of the geometry sub_tree*/
            g_ptr = rg_ptr.g_link;
            alter ( 2, g_ptr, att1, "value");
            shape ( g_ptr, att2);
            /* Use shape UPO, to append the appropriate
            geometry subtree to the region descriptor*/
    else if class_id = 21 or 22
        g_ptr = locate (region_name, 2, att1);
        alter (2, g_ptr, "value", att2);
    else if class_id > 30 and class_id < 39
        p_ptr = rg_ptr.p_link;
        p_att = 30 - class_id;
        alter (3, p_ptr, p_att, att2);
end
```

The procedure *get_all_fields()* is an I/O operation that reads one record at a time from the specified file and "chops" each record into the appropriate number of fields.

2. Device operations : The description of a semiconductor device as per figure (4.6), lacks various information describing global characteristics of the device as a whole, such as, the technology used to build the device. In our earlier attempt to provide such information, we tried creating a node that would act as a root for all region descriptor nodes. This proved to be impractical and would complicate the device description; moreover such a structure might limit the total number of regions of the device. As a result we have defined the global device information in a "Device Cluster", as shown in figure (5.5). The cluster is built of the previously defined classes of nodes, namely a region descriptor, where the *value* attribute is set to "DEVICE", and the *Name* attribute is user defined. At the g_link a geometry descriptor node is linked and the *type* attribute is instantiated as "DEV-

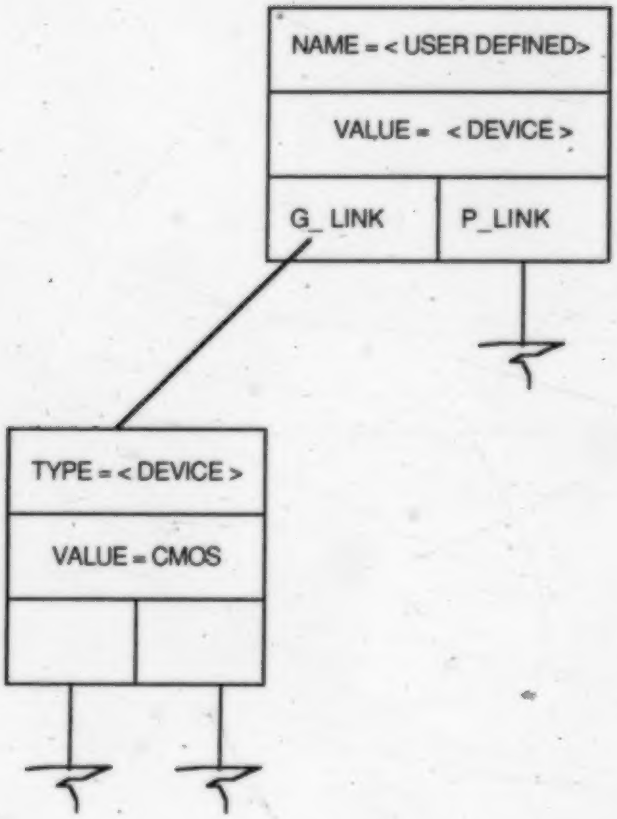

NAME = < USER DEFINED>
VALUE = < DEVICE >
G_ LINK
P_LINK
TYPE = < DEVICE >
VALUE = CMOS

Figure (5.4) : Device cluster

ICE"; the *value* attribute is again user defined, e.g. CMOS, Bipolar ...etc.

From the user point of view, the device is not just the cluster but the sum description of all regions. At a given instant the user might wish to apply an appropriate UPO to the device as a whole. We therefore have allowed the user to apply *delete, list, save,* and *load* to the device as a whole. In this case the user will provide a "wild card" (*) instead of a region_name. The system will then substitute *browse* for *v_browse* as the node referencing mechanism to propagate through the device as a whole including the "device cluster".

5.5. Tree Building Mechanisms (TBM) :

Considering the previous functions and mechanisms, we note the importance of having a simple and consistent tree building mechanism that would utilize the SPO operations and serve the UPO operations.

Considering the object model application and figures (5.1) and (5.2) which illustrate the geometry trees, we note that they essentially could be classified as follows :

1. Defined depth trees : Referring to figure (5.1) representing generic objects such as a circle, rectangle, ellipse, and pocket respectively, we note that the tree structures are well defined in terms of node types and tree depth, which are two levels at the most.
2. Undefined depth trees : Similar inspection of figure (5.2) representing iterative objects such as curvatures and polygons reveals well defined node types. In this case, the depth of the tree is undefined as the number of polygon vertices or the polynomial terms representing the curvature are not pre-defined by the system. These terms are expandable to the limits of the system memory.

Therefore, invoking the *shape* UPO, will require two separate mechanisms of tree creation, namely "Generic Tree Building Mechanism" and "Iterative Tree Building Mechanism".

5.5.1. Generic Tree Building Mechanism :

This mechanism is used to construct "Defined depth trees", and is implemented using the following algorithm :

```
procedure build_gener_tree ( Form : @char, root: geo_pointer)
const gpar_class = 1 ;
var first_type : char ;
begin
    if Form = 'Circle' ;
    then build_circle ( root);
    else
        first_type = get_node_type (Form);
        build_rpe (root, first_type );
    end
end

procedure build_circle( root : geo_pointer)
begin
    root.r_child = allocate ( gpar_class, "xcr" , NULL);
    root.l_child = allocate ( gpar_class, "ycr" , NULL);
    root = root.r_child ;
    root.r_child = allocate ( gpar_class, "rd" , NULL);
end

procedure build_rpe ( root : geo_pointer , type : char )
    var next_root : geo_pointer ;
        second_type : char;
        third_type : char ;
        fourth_type : char ;
    begin
        second_type  = get_next_type( type);
        third_type  = get_next_type( second_type);
        fourth_type  = get_next_type(third_type);
        root.l_child = allocate ( gpar_class , type , NULL);
        root.r_child = allocate ( gpar_class , second_type , NULL);
        next_root = root.l_child;
        next_root.l_child = allocate ( gpar_class , third_type , NULL);
        next_root = root.r_child;
        next_root.r_child = allocate ( gpar_class , fourth_type , NULL);
end
```

The algorithm is explained as follows :

a. The procedure is called from *shape* where its arguments *Form* pointing to the geometric form, and a pointer to *root,* the class_2 node to which the sub-tree is to be appended.

b. If the geometric form is a circle, pass the root pointer to the *build_circle* procedure. This tree is a special form as per figure (5.4).

c. As the other geometric forms handled by the mechanism are identical in structure but differ in the type of nodes, the *get_node_type* function will return the *first_type,* i.e. the type of the first left_child of the tree describing the "Form".

d. *First_type* and *root_pointer* are passed to the general tree building mechanism *rpe_build* (Rectangle_Pocket_Ellipse builder). The remaining types of nodes, see figure (5.1), are returned from *get_next_type().*

e. Using *allocate* the four nodes are created then linked.

5.5.2. Iterative Tree Building Mechanism :

This mechanism is used to build trees of "Undefined Depth". According to the above definition this mechanism is invoked during the user's instantiation of the tree. Thus, during instantiation the user will iteratively define the tree depth to the system. For these reasons, it is more appropriate to invoke this mechanism through the *ishape*, instead of the *shape* UPO.

In light of this development, *ishape* is modified to pass the root pointer and geometric form to the iterative building mechanism, as follows :

```
procedure ishape (region_name )
var rg_ptr : region_pointer;
    geo_ptr : geo_pointer;
    destination : function_pointer;
begin
    rg_ptr = locate (region_name, 0, NULL );
    /*Find the named region*/
    geo_ptr = rg_ptr.g_link;
    if geo_ptr.value = 'Polygon'  or 'Curve'
    then build_iter_tree (geo_ptr.value , geo_ptr);
    else
    destination = @in_out();
    v_browse (destination, geo_ptr);
end
```

The following algorithm implements the iterative tree building mechanism :

```
procedure build_iter_tree ( geo_form : @char; root : geo_pointer)
var l_parent: geo_pointer;
    r_parent: geo_pointer;
    new_node: geo_pointer;
    buffer : @char;
    user_need : boolean;
    node_type : @char;
begin
    l_parent = r_parent = root;
    user_need = true;
    first_type = get_node_type(geo_form);
    second_type =  get_node_type(first_type);
    while user_need true;
    do
        buffer = get_user_input();
        if buffer = NULL;
        then user_need = false;
        else node_type = first_type;
        new_node = allocate (1, node_type, NULL);
        alter (1, new_node, "value", buffer);
        l_parent.l_child = new_node;
        l_parent = new_node;
        buffer = get_user_input();
        node_type = second_type;
        new_node = allocate (1, node_type, NULL);
        alter (1, new_node, "value", buffer);
        r_parent.r_child = new_node;
        r_parent = new_node;
end
```

5.6. Summary and Illustration of Concepts :

In the previous sections we have defined the System Primitive Operations SPOs, and their algorithmic combinations which provide higher order constructs defined as the User Primitive Operations UPOs. These two sets of operations provide two levels of interaction, which are the stages through which a user will manipulate the simulated device. To integrate the discussion of mechanisms, procedures and operations of this chapter, refer to figure (5.5). This figure illustrates user-object levels of interaction. The user is being offered a number of operations to manipulate the device either as a whole or each region separately. This is the first level of interaction, where the user has access to the "User Primitive Operations", namely, *create, name, type, delete, copy, list, shape, ishape, ppar, load,* and *save.* The user interacts with this level through a graphical layer. Associated with the "UPO"s are the external and auxiliary modules. The external modules are the mesh generator, PDE solver and the graphics package. The auxiliary modules are mainly system maintenance operations. As previously defined "UPO"s are algorithmic constructs of the SPO simpler operations, namely, *alter, transcribe, erase,* and *pres_ive.* These operations are linked through the construction mechanisms, *locate,* and *allocate* of the Node Referencing Mechanism "NRM" and *h_browse* and *v_browse* of the tree browsing mechanism "TBM".

Figure (5.6) illustrates the corresponding operational model of the system. This view of the system identifies four main tasks a user performs during a simulation session. These tasks are specification, manipulation, solving and system maintenance. The specification of the device is done on region-by-region basis. A user will manipulate

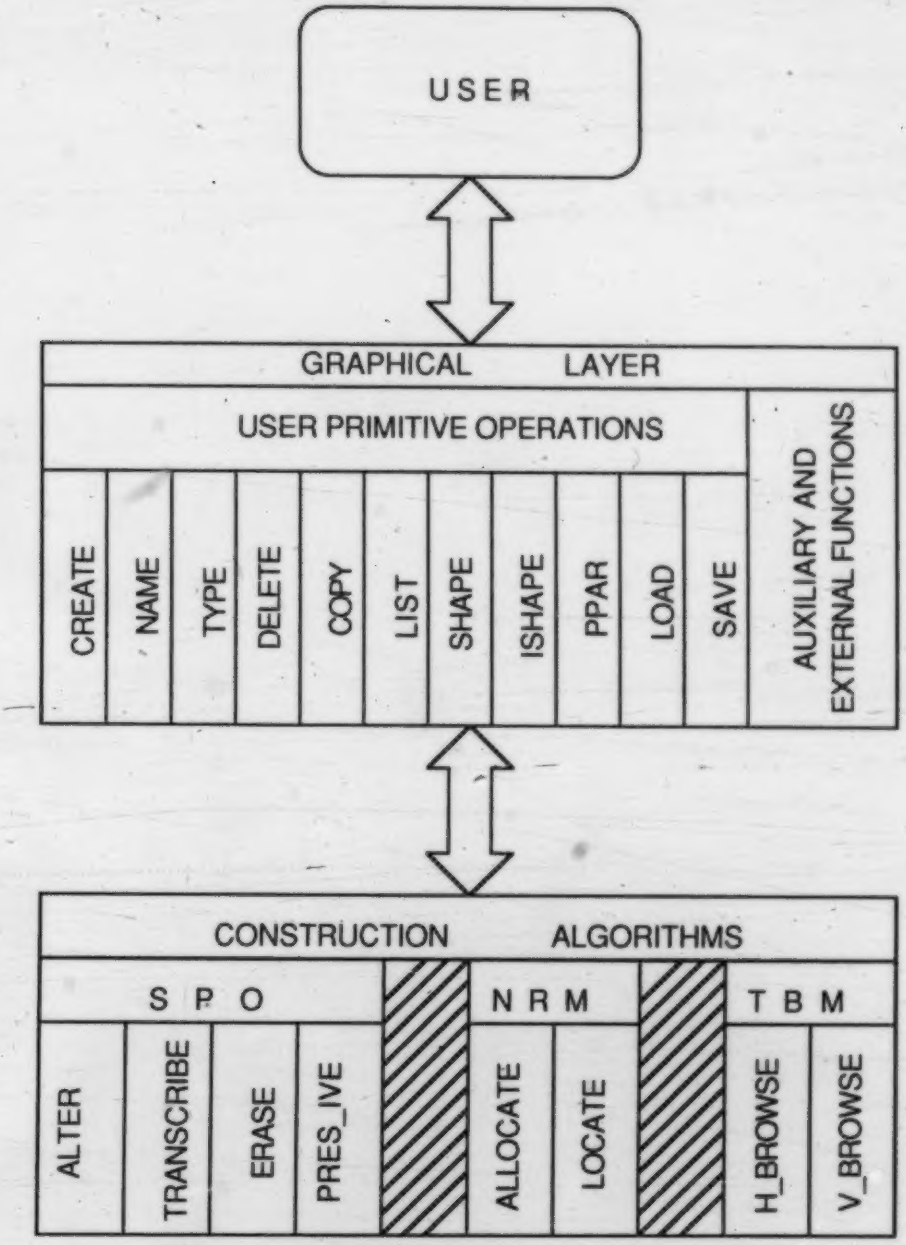

USER
GRAPHICAL LAYER
USER PRIMITIVE OPERATIONS
CREATE
NAME
TYPE
DELETE
COPY
LIST
SHAPE
ISHAPE
PPAR
LOAD
SAVE
AUXILIARY AND EXTERNAL FUNCTIONS
CONSTRUCTION ALGORITHMS
SPO
ALTER
TRANSCRIBE
ERASE
PRES_IVE
NRM
ALLOCATE
LOCATE
TBM
H_BROWSE
V_BROWSE

Figure (5.5) : Albertina's levels of interaction.

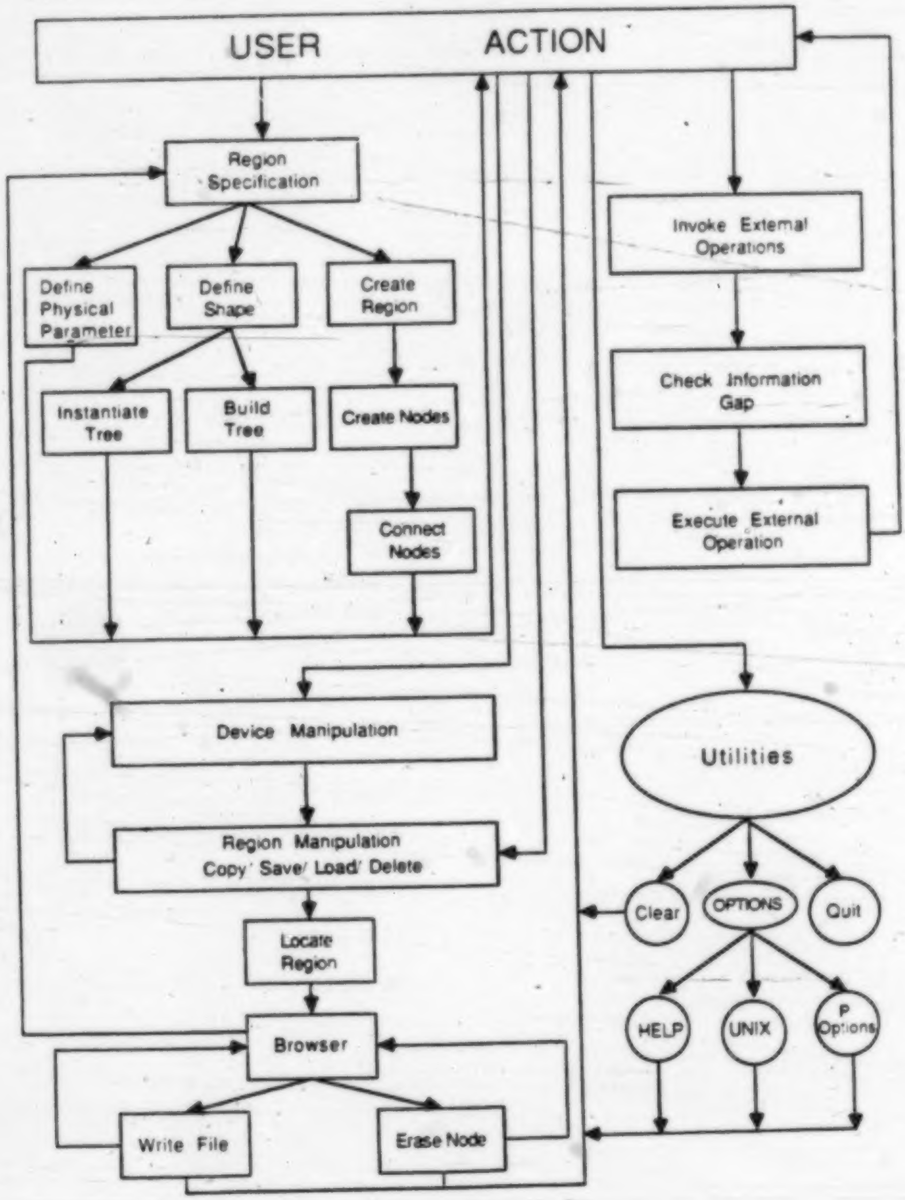

USER ACTION
Region Specification
Define Physical Parameter
Define Shape
Create Region
Instantiate Tree
Build Tree
Create Nodes
Connect Nodes
Invoke External Operations
Check Information Gap
Execute External Operation
Device Manipulation
Region Manipulation Copy/ Save/ Load/ Delete
Locate Region
Browser
Write File
Erase Node
Utilities
Clear
OPTIONS
Quit
HELP
UNIX
P Options

Figure (5.6) : Albertina's operational model.

either one region or the device as a whole. Solving is basically invoking the solution steps i.e. the mesh generator and the PDE solver. System maintenance is classified as utilities, where the user can reorganize the graphical layer, execute some options or exit. Any of the above tasks is executed by the *user action*, which is a result of the user interaction with the system's graphical layer. Once the task has been executed control is returned to the user. These operations are further divided into sub-operations and are explained with reference to figure (5.6) as follows :

1. Region specification operations : This group consists of the *Region operations* discussed in section (5.4.1), where regions are created and their geometry and physics are defined. As a result of further user action interpretation a sub-operation is selected. The sub-operations are :
    a. Create region : Creating a region is implemented according to the discussion of this chapter and section (4.3). Therefore, this sub-operation will create one descriptor node of each class and link them according to the tree description of a region.
    b. Define shape : This sub-operation is responsible for either creating the geometry sub-tree of a region or the instantiation of a sub-tree that has already been built. Creating the sub-tree is done automatically by the system, while instantiation requires further user interaction. Through the interaction the user will enter the instantiation values.
    c. Define physical parameters : This sub-operation is responsible for instantiating an attribute of the region's physical descriptor node. The instantiated value of this attribute is an argument of the user action.
2. Device manipulation operations : The user access to the simulated device as a whole is the responsibility of this group. A *user action* is interpreted by the system and implemented. Such actions may iteratively use *Region Specification* operations. For example in the case of a *copy* operation, if the **to_region** does not exist the system will create it using the *create_region* sub-operation. In any case,

the device manipulation operations will utilize the browsing mechanisms *h_browse* and *v_browse* in order to locate a region and acquire its data. The data could then be either used (e.g. written to file) or erased. Depending on the user action an iteration of the browsing mechanisms could be implemented (e.g. delete a whole region).

3. External modules operations : The user interface provides the user with means to invoke the second and third solution steps, i.e. mesh generator and PDE solver. This group of operations is responsible for these tasks. Prior to invoking any of the external modules a check is run to detect any information gaps. For example, to ensure that all the geometry sub-trees are instantiated prior to invoking the mesh generator, or the existence of the mesh data prior to invoking the solver. These modules are separate programs that could be concurrently executed. Therefore, once the check is successful, the module is invoked and control is returned to the user. Further details of invoking and monitoring these modules is provided in chapter 6.

4. Utilities : A variety of tasks are performed through this group, which is mainly concerned with system and user data maintenance. Examples of such sub-operations are :

   a. Clear : This sub-operation allows the user to clear the screen, i.e. erase the data accumulated by the graphical layer on the terminal screen.

   b. Quit : The user will *exit* the simulation session by using this sub-operation. The geometric and physical description of the device are automatically saved prior to termination of the session.

   c. Options : This group of sub-operations are mainly options the user can use to perform a variety of operations. It also provides the designer means to add any future expansion. For example :

      i. The expansion of the physical parameter operation *options* (see section

6.5.3.) is part of this sub-operation.

ii. On line HELP, which provides the user with an explanation of the system's various operations and models.

iii. Means for the user to *escape* into the operating system, Unix.

6. System Integration

In section (2.3) we described the steps involved in a semiconductor device simulation session. The previous chapters discussed the operations and mechanisms Albertina provides to accomplish the first step of describing the device. We therefore can rearrange the simulation steps as follows :

1. Geometry Definition : Defining the topography of the simulated device is a combination of two steps :
   a. Shape Assignment : Using the *shape* UPO, each region is assigned a particular geometric form.
   b. Shape Instantiation : Each parameter of a geometric form is instantiated a numeric value; this step is achieved by the *Ishape* UPO.
2. Physical Definition : This step is responsible for defining another view of each region. Through the *Ppar* User Primitive Operation, the user can select a model and assign it to an attribute of the region's physical descriptor node.
3. Invoking the "External Modules": As previously described, the "Mesh Generator" and the "PDE Solver" are seen, with respect to Albertina's new structure, as external modules. Albertina's new structure has transferred the responsibility of invoking and monitoring these modules to the system. This required several mechanisms, which includes a mechanism to calculate numerical values for the different physical parameters.

In this chapter we shall discuss our implementation of the "Graphical Layer" through which all UPOs are offered to the user in the form of a *panel*, see figure (4.2). We then proceed to discuss the method through which the new structure of Albertina provides numerical values for the physical parameters. This is followed by a description of mechanisms utilized to invoke and monitor the mesh generator and the PDE solver. Finally we discuss the graphics package provided with Albertina.

6.1. Application of Albertina's Graphical Layer :

The graphical layer is the frontier through which the user interacts with all aspects of the simulation session. As mentioned in section (4.4), this part of the user interface is hardware and software dependent. In this section we provide application dependent details of the graphical layer.

The user interface has been implemented in a Unix environment, on a Sun 3/160 workstation, with a high resolution color graphic terminal. "C" [62] is used as the programming language. The graphical layer implementation utilizes *Sunview* [63, 64] an acronym for, Sun Visual Integrated Environment for Workstations, defined as : A system to support applications with interactive, graphics-based interfaces running within windows. *Sunview* consists of two major areas of functionality : building blocks for output, and a run-time system for managing input. Building blocks are *windows* which could be of several types. The run time system is based on a *Notifier* which distributes user input from a certain *window* to the appropriate processer. The *window manager* will handle overlapping windows. Graphics within *Sunview* are an implementation of GKS [65], CGI [66] and CORE [67], that are discussed in [68]. *Windows* are built in a hierarchy, which could be viewed as a tree of windows in which the non-leaf nodes are frames and the leaf nodes are subwindows.

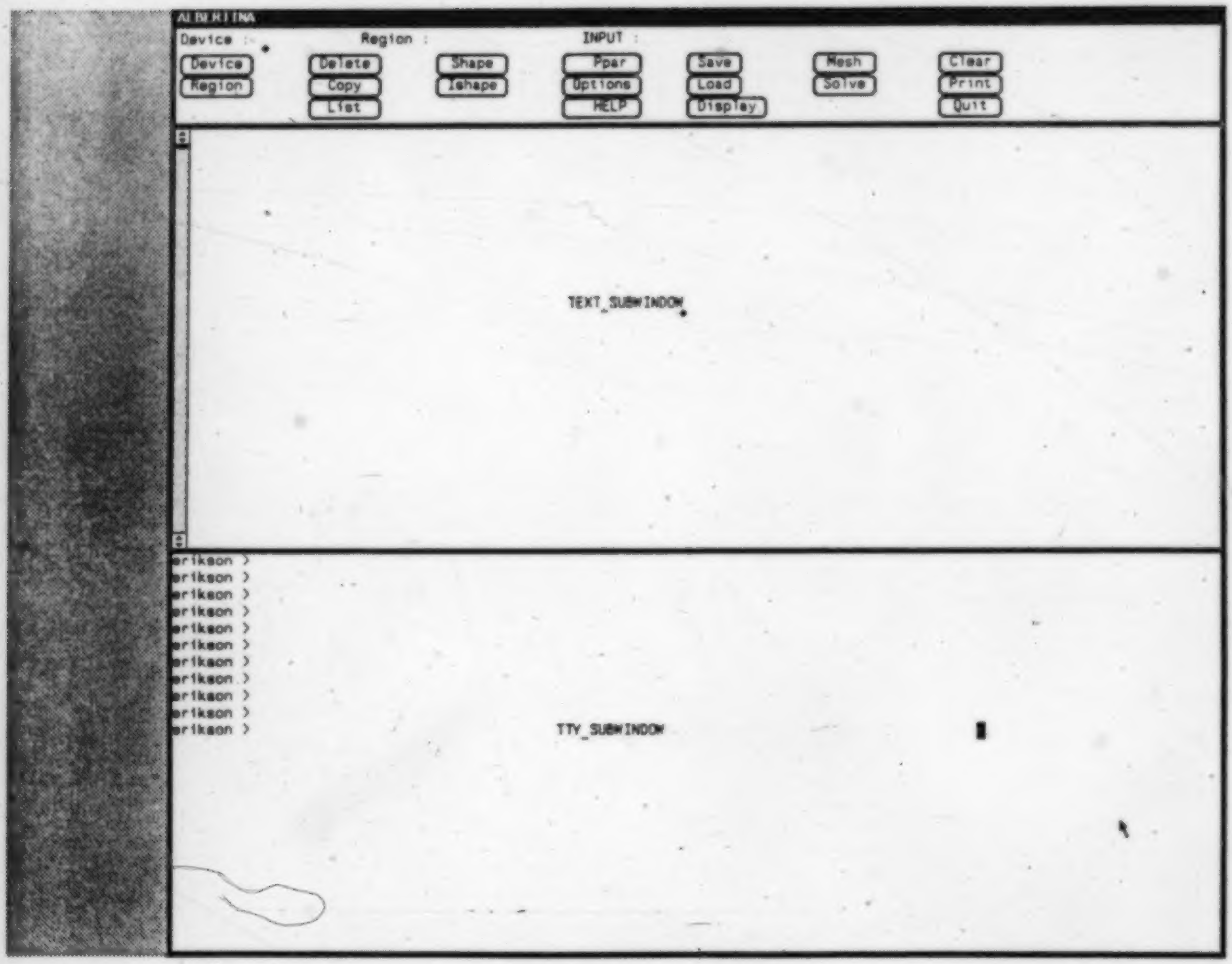
Sunview provides the facilities of *panels, menus,* and the graphical capabilities, thus, accommodating our specification of the "Graphical Layer" in section (4.1).

6.1.1. General Layout :

Figure (6.1) illustrates the general layout of the graphical layer as seen by the user on the terminal. In this figure a *window* consists the of main boundaries of the user interface. A window is created as a *frame* through the *Sunview* system instruction [64] :

frame = window_create (Parent_window, Window_type, window_att, 0)

The user interface main window, henceforth called *base_frame* is created as:


ALBERTINA
Device :
Region :
INPUT :
Device
Region
Delete
Copy
List
Shape
Ishape
Ppar
Options
HELP
Save
Load
Display
Mesh
Solve
Clear
Print
Quit
TEXT_SUBWINDOW
TTY_SUBWINDOW

```
erikson >
erikson >
erikson >
erikson >
erikson >
erikson >
erikson >
erikson >
erikson >
erikson >
erikson >
```

Figure (6.1) : Application of Albertina's graphical layer.

base_frame = window_create (NULL, FRAME, FRAME_ATT, 0)

The first argument indicates that this is a root window with no parent; *FRAME* indicates the type of the window; *FRAME_ATT* is a collection of attributes pertaining to the window (e.g. window dimensions). Further subwindows are created within the *base_frame* as leaves.

Albertina's graphical layer is grouped into three major sections :

1. Panels section : The top section is a collection of panels, where a panel is defined as : A subwidow containing items through which the user interacts with a program. In our implementation, panels will enable the user to select any action within the following categories :
    a. A "User Primitive Operation", (e.g. *shape, Ishape).*
    b. An "External Module" , (e.g. *Mesh).*
    c. "System Maintenance" procedure, (e.g. *Clear)*
    d. An "Auxiliary function" invoker, (e.g. *Help, Options).*)

The function of each of the above panels is invoked through a *Locator,* which is a three-button "Mouse". Data are entered through the alphanumeric keyboard.

Once a panel is selected, a variety of actions could take place, as illustrated in figure (6.2). The panel could directly invoke an action (e.g. *delete, list),* or proceed through a hierarchy of menus, providing one or more arguments to the "Action Process". The menu hierarchy could be of one level (e.g. *shape, clear),* or could proceed to a second level (e.g. *Ppar, Region).*

Panels are created in the same manner as the *base_frame* was created :

panel = window_create (base_frame, PANEL, PANEL_ATT, 0)

The first argument indicates that the created panel is a subwindow within *base_frame* and the new subwindow is of type *PANEL.* The *PANEL_ATT* argument carries information pertaining to the panel such as the panel's physical location within

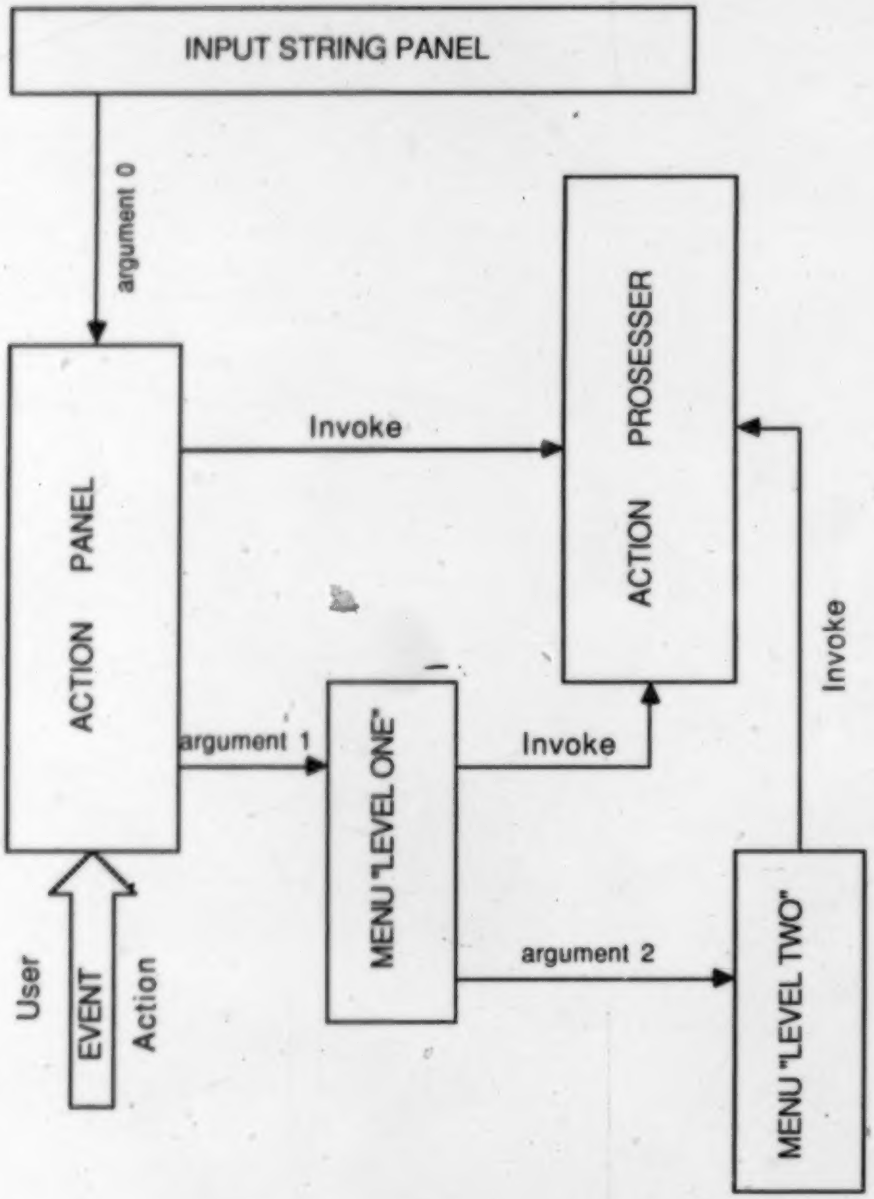

INPUT STRING PANEL
argument 0
ACTION PANEL
Invoke
ACTION PROSESSER
argument 1
MENU "LEVEL ONE"
Invoke
Invoke
argument 2
MENU "LEVEL TWO"
User
EVENT
Action

Figure (6.2) : Action-Panel-Menu relationship the *base_frame*, the possible *Menus* the panel handles, and the final *Action function* invoked once the panel is selected.

2. Text section : The middle section of the screen is of type *Text subwindow*. The basic function of this type is to interact with the user to display and edit a sequence of ASCII characters. These characters could be stored either in a file or in primary memory. From the programmer's point of view, a text subwindow is an opaque object upon which a set of operations can be performed. Through this section certain interactive procedures (e.g. *list, Ishape),* between the user and the system take place. The *Text subwindow* is created the same way as a *panel*, with some variations to the attributes as follows :

   txt_sw = window_create (base_frame, TEXTSW, TXT_ATT, 0).

3. Tty section: The last section on the screen is a standard tty emulator, emulating a SUN terminal. A Unix sub-shell exists in this section as a subwindow, allowing the user to escape into the Unix operating system. The escape is either to manipulate or inspect a process which could be a non-Albertina process. Also some Albertina procedures are run through this window due to its input-output capabilities.

Detailed discussion of the creation, manipulation and attributes of the above subwindows is found in [63, 64].

6.1.2. Input-Output Considerations :

Input and Output are again implementation specific aspects of this work. Using the Sunview input paradigm, any user action (input) is classified as an *EVENT*. An event is classified according to a numerical value called *identifying code,* which is returned from the Sunview process *event_id()* as follows :

a. An ASCII event : Where the returned value is in the range [0 .. 255]. Any value within this range indicates that the event is a keyboard action.

b. Locator button event : If the standard Sun locator (Mouse) is pressed, a value in the range [32530 .. 32546] inclusive is returned indicating one of the locator's three buttons.

c. Locator motion event : Once the mouse is moved, a value in the range [32512 .. 32525] is returned.

According to the above definitions and classifications, we discuss the input and output aspects of the different sections of the user interface:

1. Panels : Reviewing figure (6.2), we note that the "User Action" corresponds to "Locator Button Event". When the user activates the appropriate "Mouse Button", the action designated to the panel is invoked. This could be a direct invokement of an "Action Processor" or the display of menu(s). The designation of the action processer, henceforth called *event_proc()*, is done during the creation of the panel through an attribute denoted as *PAN_ATT*.

2. Text subwindows : Selecting a panel might result in invoking an *event_proc()*, which in turn will either display an output on the textsubwindow or require an input from it. Sunview provides a *Notifier* mechanism which is basically an event-driven dispatcher. *Notifier* will distribute events to a collection of *clients (processes)*. The notifier will detect events and dispatch them to a client that has previously "registered" with it. Client registration means establishing an interest in a certain type of event, and is dynamically implemented via the following two Sunview system functions :

   notify_interpose_event_func (subwindow_type, client, when),

   notify_remove_event_func (subwindow_type, client, when).

   The first system call registers the client to the *Notifier;* the second removes the registration. In this situation *subwindow_type* is the type of subwindow from which the event detection is required, *client* is the process to which the event is to be dispatched, and *when* is to advise the *Notifier* when to dispatch the event.

Relating the above mechanisms to the "User Primitive Operations", we review the algorithm implementing *Ishape* in which we find the statement :

buffer = get_user_input();

This statment indicates that *get_user_input()* is an input function returning a character pointer (buffer). The buffer contains the user input. Since the input occurs within the textsubwindow, this function will conform to the above I/O mechanism. Hence, *get_user_input()* is the forementioned *client*, and has to be registered with the *Notifier* upon invoking *Ishape* as follows :

notify_interpose_event_func (text_subwindow, get_user_input(), ALL)

The following algorithm implements *get_user_input()*:

```
type input_buffer = array [1..10] of char

function get_user_input ( event : char ) @char;

const default_value = 0.0;

var buffer : input_buffer;
    no_of_chars : integer;   /*Number of characters in the buffer */
    begin
        no_of_chars   = 0 ;
        if event = ascii;
            then if event = RETURN;
            /* User did hit "return" on the keyboard signaling end of input */
                then if no_of_chars = 0;
                    then buffer = default_value;
                    notify_remove_event_func(text_subwindow,  get_user_input(),
                    ALL)
                else
                    notify_remove_event_func(text_subwindow,  get_user_input(),
                ALL)
        else
                buffer [no_of_chars] = event;
                no_of_chars = no_of_chars + 1;
        else ignore_input();
end
```

The following truth table might simplify the procedure, by declaring :

α = TRUE if event = ascii character else α = FALSE;

β = TRUE if event = "Return" character else β = FALSE;

γ = TRUE if number of characters in buffer = 0 else γ = FALSE;

X = don't care state.

α	β	γ	Action
FALSE	X	X	ignore event
TRUE	FALSE	X	add event to buffer
TRUE	TRUE	FALSE	return buffer
TRUE	TRUE	TRUE	buffer = default_value, return buffer.

Table (6.1) : I/O mechanism truth table.

The only problem that we might face using this algorithm is the re-initialization of the variable *no_of_chars* upon re-entry to the *event_proc();* to avoid such a hazard, *no_of_chars* is declared as a global variable and initialized before removing the function's registration.

3. Tty subwindow : Input and Output from this subwindow is simpler than the previous two, as it runs a sub-shell. It therefore utilizes the UNIX system calls of standard Input/Output library, such as *printf, scanf, fgets ...etc.*

Further detailed discussion of the *Notifier* mechanism is found in [58].

6.2. Definition of the Physical Parameters System Calls :

As we mentioned in the introduction of this chapter, the new structure of Albertina needs a mechanism through which numerical values for the user selected physical models are provided to the external modules. During the course of developing this work many mechanisms were considered. The final result was the creation of the "Physical Parameters System Calls" (PPSC). This is defined as : A library of functions, each returning the numerical value of a certain physical parameter according to a user designated model. The functions are independent subprograms that are compiled separately from all other programs forming Albertina. Each PPCS is invoked through a module *alba_pipe* inserted in the caller program. This module will invoke the appropriate PPSC and pass to it a set of **arguments** containing data. The data is used by the PPSC to calculate the numerical value of a certain parameter; *alba_pipe* will then read the final result from the PPSC and return it to the caller.

In figure (4.1), we have defined the physical package, which so far has included "User Primitive Operations" such as *Ppar,* and the library of "Physical Models". Now we incorporate the PPSC, as the third part in this package. In the following sections we shall discuss the mechanisms through which *alba_pipe* invokes the different PPSCs; we then proceed to define each PPSC and explain its calculation methods.

6.2.1. Invokement of PPSCs:

The abovementioned module *alba_pipe* invokes any PPSC through an interprocess channel *pipe* created by the Unix system call *popen* [69, 70]. The system call invokes the appropriate PPSC and establishes a data channel between itself and the caller program. The arguments to *popen* are pointers to null-terminated strings indicating the relevant PPSC, and an input-output mode. In our case, the I/O mode is **read** , allowing *alba_pipe* to read the final result from the PPSC and return it to the caller program. *Alba_pipe* is henceforth treated as a function and is called as follows:

ppar_value = alba_pipe (pparam, region_name, arg1, arg2,.........)

```
    ppar_value : real;  /* The returned value of a physical parameter */
    pparam : @char;     /* The physical parameter to be calculated */
    region_name : @char;
    /*The name of the region upon which PPCS will operate */
    arg1, arg2,......: real;
    /* Numerical values of the parameters pertaining to the model of the
    physical parameter */
```

The invoker module *alba_pipe* is implemented as follows :

```
function alba_pipe (pparam: @char, region_name: @char, arg1, arg2, arg3...: real)
real;
begin
    if param = valid_ppsc; /* Check the validity (existence) of the PPSC */
        then alba_pipe = create_pipe (param, arg1, arg2, arg3 ...);
        /* Create the pipe and read the result */
    else
        print_error (); /* Advise caller of error */
        alba_pipe = 0;
end
```

Reviewing the definition of *popen* we note that its arguments are null-terminated character pointers. This creates a discrepancy with the above algorithm where real numbers are used. This problem is overcome by using the output conversion C library function *ecvt* [69] which will convert a double precision number to a null-terminated string. Figure (6.3) illustrates the above discussion, where *alba_pipe* is inserted within the caller program. Once *alba_pipe* is called, an I/O pipe is created to a certain function in the PPSC library. The PPSC will calculate the value of the parameter. This value is returned to *alba_pipe* which communicates it to the caller program.

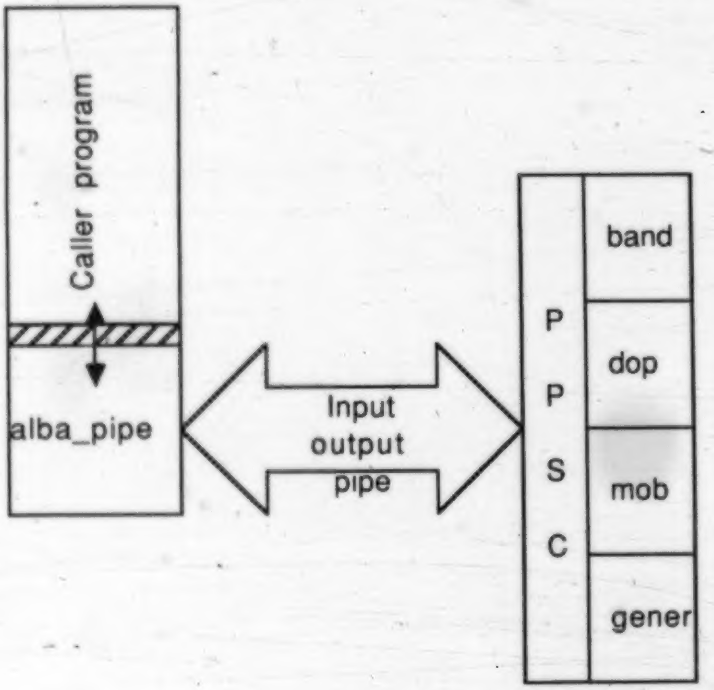

Caller program
alba_pipe
Input
output
pipe
P
P
S
C
band
dop
mob
gener

Figure (6.3) : Pipe channel to PPSC

6.3. General Application of PPSC :

According to the above discussion a separate PPSC for each physical parameter is required. Yet they are all built within the same algorithm as follows :

```
const para_number: integer;
function ppcs_* (region_name : @char ; arguments..: real) real;
var model: @char;
begin
    model = get_ppar (region_name, param_number);
    case model of :
        valid_model: ppsc_* = calculate_value( arguments...);
end --
```

In this algorithm the ppsc_* is any of the PPSC library functions. *Get_ppars* is an extraction function which returns the model that was assigned by the user to a certain physical parameter. It is returned as a pointer to a character string. Hence, *get_ppars* uses two arguments; the first is *region_name* through which the region is identified and its physical descriptor node is acquired; the second argument is a constant integer *param_number* denoting the physical descriptor attribute of interest to ppsc_*. The numerical value is then obtained from another function *calculate_value* to which only the relevant arguments are passed.

6.4. Parameter Extraction :

The function *get_ppars* operates as follows :

Before invoking any of the "External Modules" i.e the mesh or PDE solver, the system dumps all physical descriptor nodes into a file. From this file *get_ppars* extracts the parameter required by the ppsc_*, hence the name of this subsection. This file is a part of a "Physical Data Pool" and is of the following format :

Header	field_1	field_2	field_3	field_4
Region Name	node class	attribute number	physical parameter	model

Table (6.2) : Format of physical parameters data pool.

The subroutine *get_ppars* extracts the assigned model by searching through the file until "Region_name" matches the given name. Next, the physical parameter (field_3) is matched with the parameter handled by the calling PPCS which is identified as *param_number* argument. The field sought is the variable residing in field_4 and is denoted by 'Model'.

6.5. Application of Physical Parameters System Calls :

In this section we shall discuss the details of each of the above PPSC functions and describe the calculation mechanisms.

6.5.1. Band Gap Narrowing PPSC (band) :

Reviewing the bandgap narrowing models discussed in section (3.6), we realize that in general the bandgap narrowing PPSC, henceforth called *band*, is called as follows :

ppar_value = alba_pipe (band, region_name, N_i*)*

region_name: @char;

N_i: real; /* Impurity concentration */

The calculations of ΔE_G models are straightforward. For example, if the user has selected the Thurmond model equation (3.105), which is a function of temperature only the region's temperature will be extracted from the physical data pool using *get_ppar*. In this case, the argument N_i will be ignored. The PPSC *band* in this case is called as follows :

ppar_value = alba_pipe (band, region_name, NULL)

6.5.2. Mobility PPSC (mob) :

The library of physical parameters provides a large number of carrier mobility models (section 5.3.2). Hence, the number of possible arguments is relatively large. The mobility PPSC *mob* is therefore called as follows :

ppar_value = alba_pipe (mob, region_name, carrier, N_i*, E ,* E_p*, x);*

where

region_name : @char;

carrier: integer; /* carrier = 0 for electron, carrier = 1 for hole */

N_i: real; /* Impurity concentration */

E: real; /*Electric filed at the referenced point V/cm */

E_p: real;

/* Electric field component normal to the direction of current flow */

x: real; /* Distance normal to the region's surface */

Similar to *band*, not all of these arguments are required by each mobility model. Therefore, the nonrelevant arguments could be declared as NULL by the caller program because *mob* will ignore it. In table (6.3), mobility models and their relevant arguments are listed. If the argument is required by a model, it is denoted by "Yes", otherwise it is marked "No".

Model (eqn #)	region_name	carrier	N_i	E	E_p	x
Scharfetter_Gummel (3.77)	Yes	Yes	No	No	No	No
Arora (3.79)	Yes	Yes	Yes	No	No	No
Caughey & Thomas (3.71)	Yes	Yes	Yes	No	No	No
Caughey & Thomas (3.72)	Yes	Yes	Yes	Yes	No	No
Thornber_Gummel (3.74)	Yes	Yes	Yes	Yes	No	No
Scharfetter_Gummel (3.78)	Yes	Yes	Yes	Yes	No	No
Yamaguchi (3.81)	Yes	Yes	Yes	Yes	Yes	No
Selberherr (3.82)	Yes	Yes	Yes	Yes	Yes	Yes

Table (6.3) : Arguments of mobility PPSC.

The mechanism for calculating the mobility μ is illustrated in figure (6.4), where the dependence of the complex models, (i.e. models encompassing more scattering mechanisms), on the simpler models is explained as follows :

When calculating a mobility value according to a certain model, we proceed upwards from the box representing the model. Consider the Selberherr surface scattering model. This model requires an initial value μ^{tg} from the "Thornber-Gummel" model, which in turn requires the following :

1. An initial value from the "Caughey and Thomas" doping dependency model.
2. The temperature dependency of v^{sat}.

These calculations are performed and returned to the "Selberherr" module (box). Using these results this module produces the final result and returns it to the caller program through the "RESULT" channel. This method is applied to the rest of the modules.

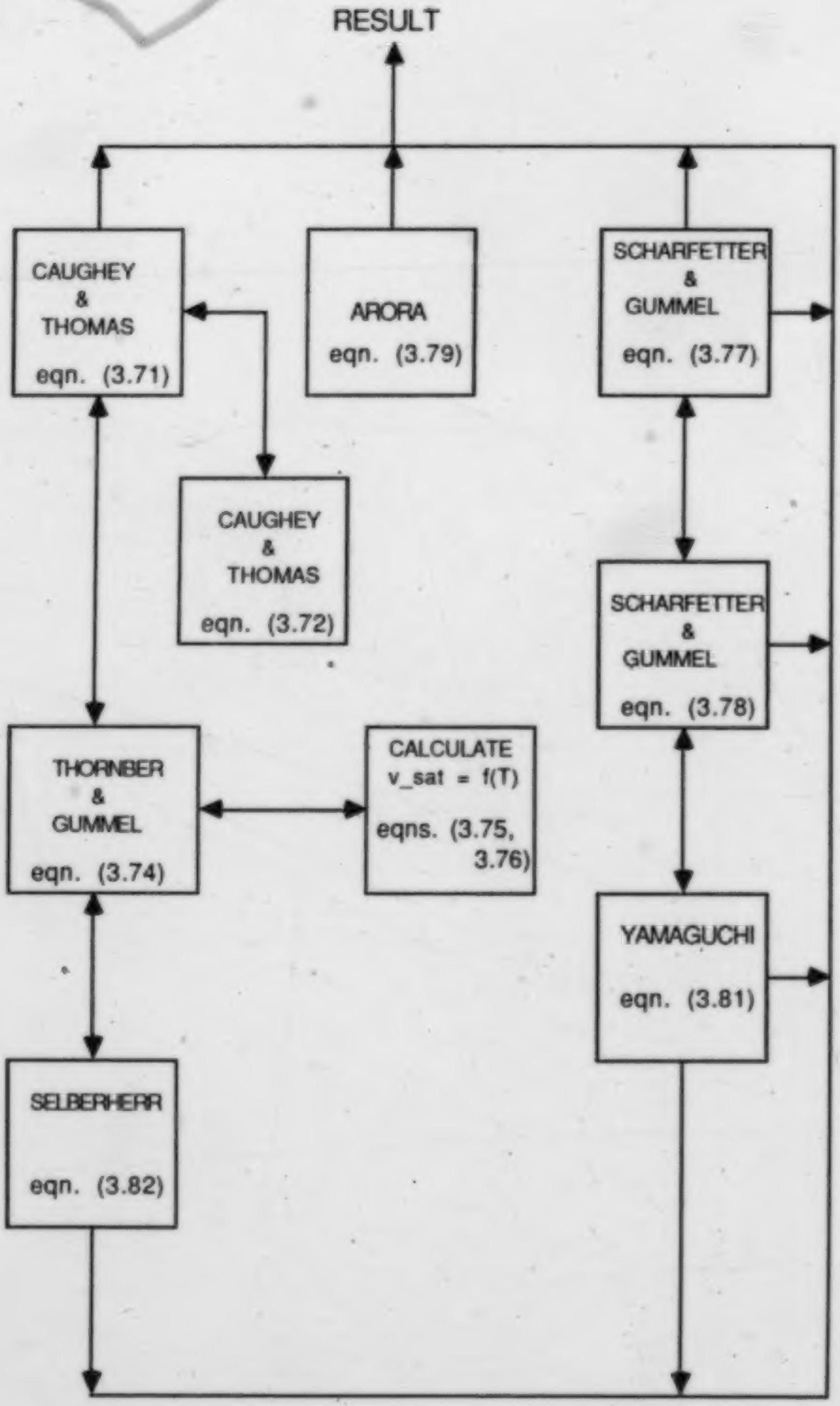

RESULT
CAUGHEY & THOMAS eqn. (3.71)
ARORA eqn. (3.79)
SCHARFETTER & GUMMEL eqn. (3.77)
CAUGHEY & THOMAS eqn. (3.72)
SCHARFETTER & GUMMEL eqn. (3.78)
THORNBER & GUMMEL eqn. (3.74)
CALCULATE v_sat = f(T) eqns. (3.75, 3.76)
YAMAGUCHI eqn. (3.81)
SELBERHERR eqn. (3.82)

Figure (6.4) : Mobility calculation

Output plots of *mob* for various models are given in figures (6.5), (6.6), (6.7) and (6.8).

6.5.3. Doping Profile PPSC (dop) :

This function named *dop* is a rather special type of PPSC. Doping profiles are usually classified as a part of a process simulation package rather than a device simulator. In Albertina we have included doping profile functions due to their importance and in order to minimize dependency on external packages. The functions are realistic models for the doping concentration in replacement of the analytical functions used by Albertina.

Reviewing section (5.2), we can summarize the variables involved in calculating the doping profile as follows :

doping_concentration = $F_d(R_p, \Delta R_p, \Delta R_l, N_m, x, y)$

where

R_p is the mean projected range.

$\Delta R_p, \Delta R_l$ are respectively the vertical and lateral standard deviation.

N_m is the implanted dose.

x, y are the lateral and vertical distances measured from the device surface.

According to equations (5.10) and (5.11) the variables R_p and ΔR_p are functions of dopant species I_s, and the implantation energy E_I. Therefore F_d is reformulated as :

doping_concentration = $F_d\,(E_I, I_s, N_m, x, y)$

These variables are provided to *dop* through one or more of the following three sources :

1. Physical descriptor node : Where the dopant species I_s and the dose N_m are attributes. Their values are acquired through *get_ppar*.
2. The caller program : Where $d_{x,y}$ are provided by the caller through *alba_pipe*, hence *dop* is called as follows :

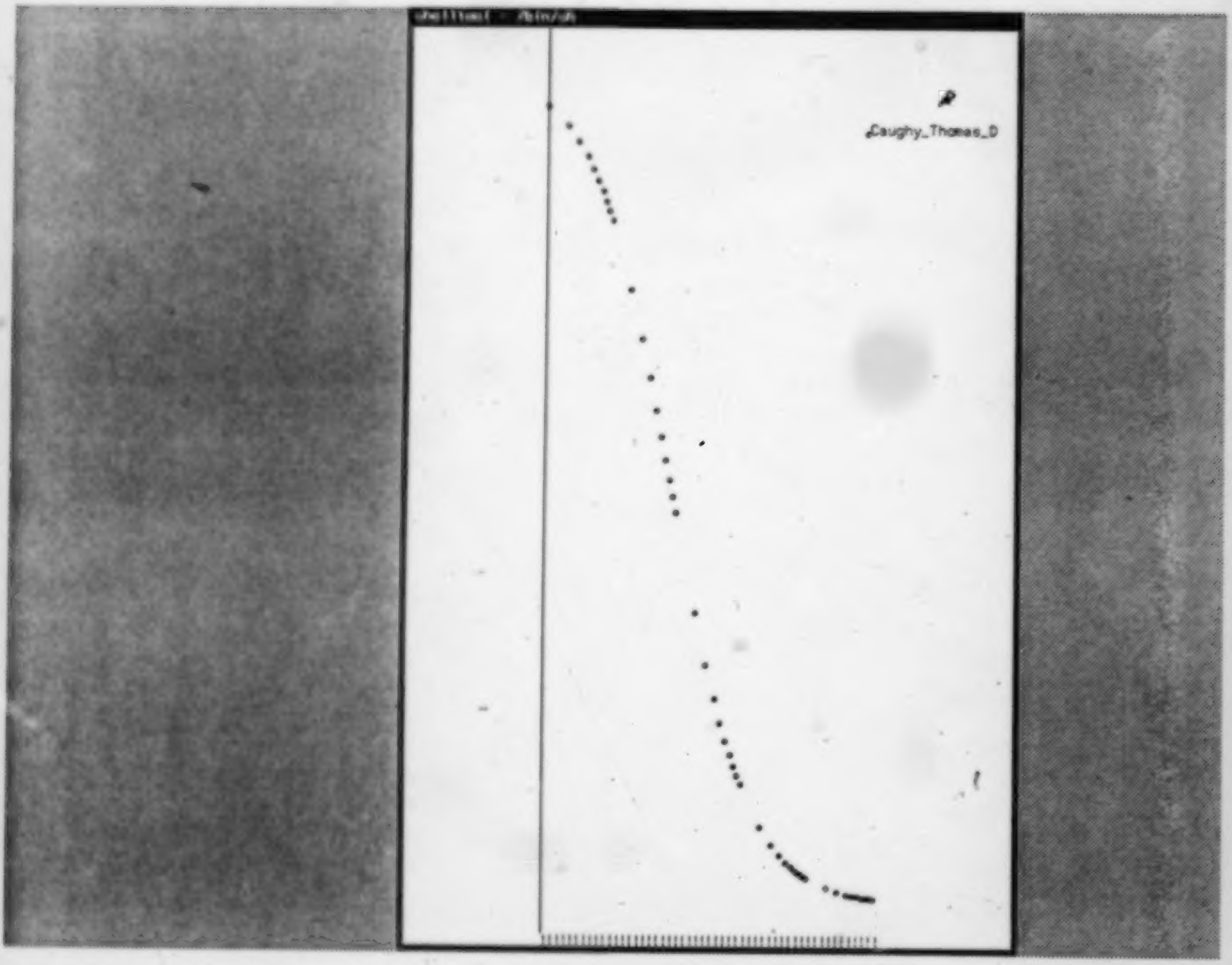


Figure (6.5) : Plot of Caughey and Thomas doping mobility model.

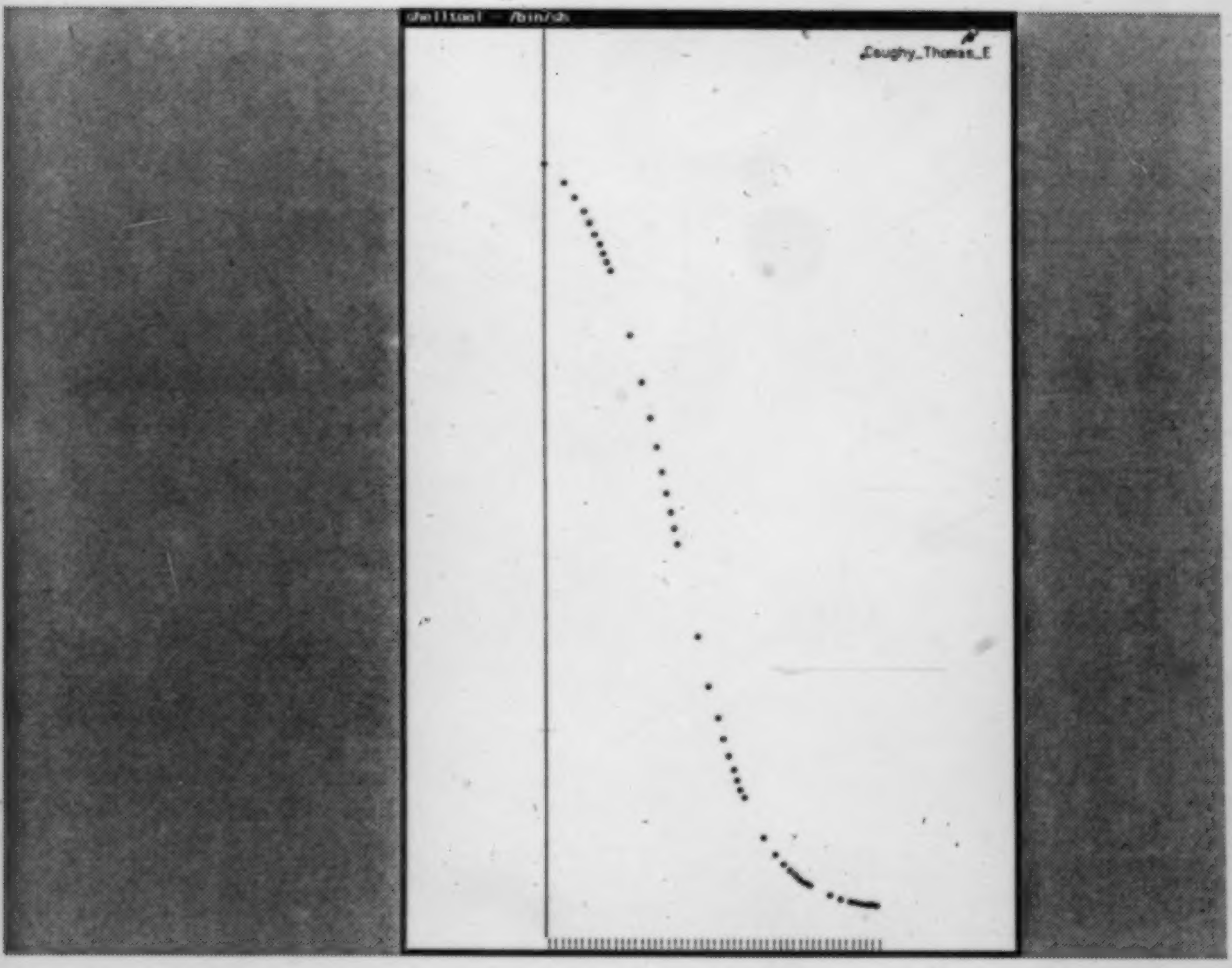


Figure (6.6) : Plot of Caughey and Thomas electric field mobility model.

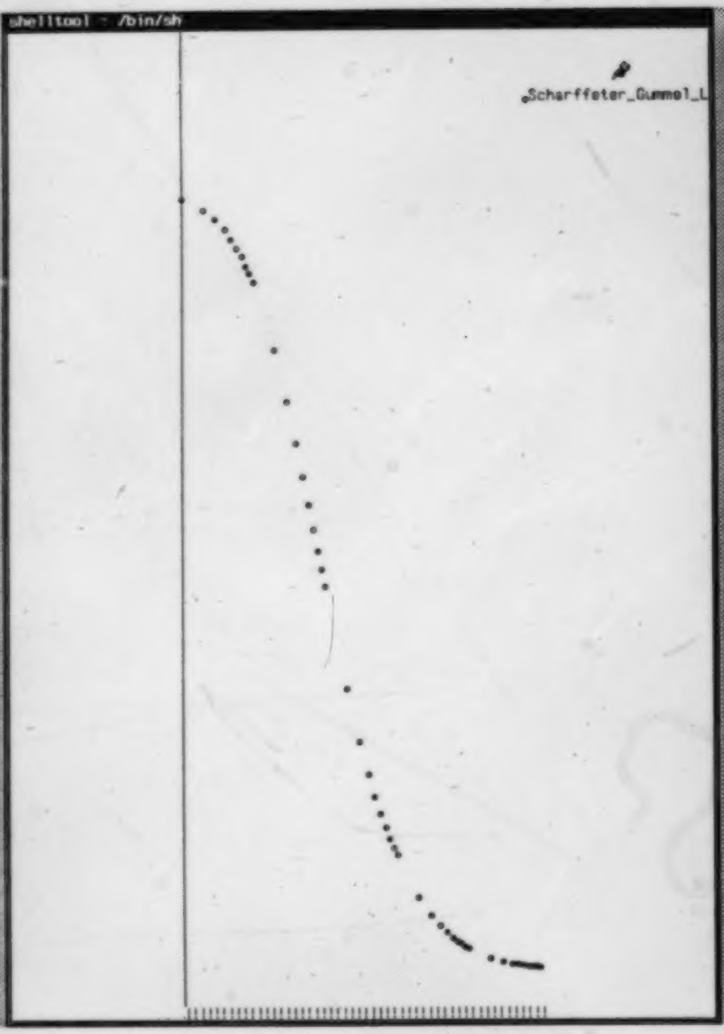


Figure (6.7) : Plot of Scharfetter & Gummel mobility model.

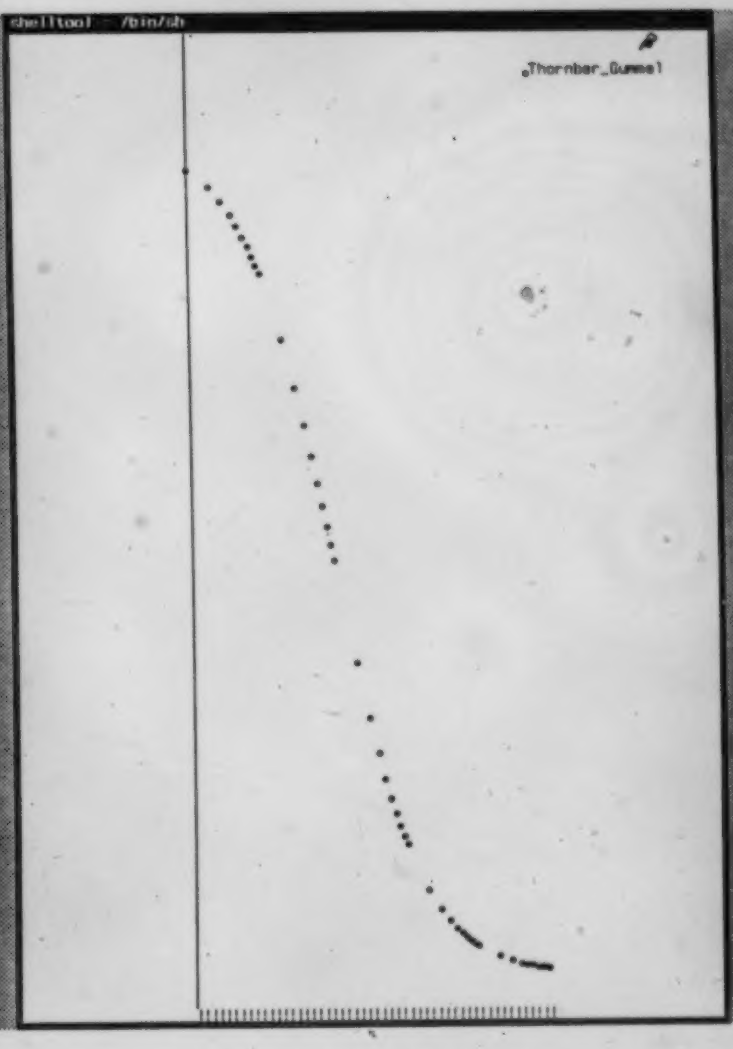


Figure (6.8) : Plot of Thornber-Gummel mobility model.

ppar_value = alba_pipe (dop, region_name, $d_{x,y}$*)*

3. OPTIONS : This mechanism is classified as an "Auxiliary function". OPTIONS is invoked from a panel (see figure 6.1), through which a user can provide values for the implantation energy E_I, R_p, ΔR_p and ΔR_l. This information is stored in another segment of the "Physical Data Pool" in the following format :

Field_1	Field_2	Field_3	Field_4
region_name	physical_parameter	model / variable	<values>

Table (6.4) : Format of OPTIONS physical data pool.

The values entered by the user and stored in the above format are acquired by *dop* through the *get_options* function. The algorithm used to implement this function is similar to that of *get_ppars*. In the algorithm implementing *dop*, *get_options* is called to inspect if the user has entered optional values for a certain parameter. If none are found they are either calculated by *dop* or set to a default value. The variables with respect to their sources are tabulated in table (6.5), where sources are listed according to their precedence. For example, in the case of E_I, first *get_options* is called, if no value is returned the a default value of 100 KeV is used.

Variable	Source
model	get_ppar
I_s	get_ppar
E_I	get-options/default = 100 KeV
R_p	get_options/proj_dev
ΔR_p	get_options/proj_dev
dose	get_par
x	caller provided
y	caller provided
a	caller provided
ΔR_l	get_option/default
γ	get_option/default
β_p	get_option/calculated $\beta_p = \beta_{\min}$

Table (6.5) : Sources of doping parameters.

The variable β_p is calculated as $\beta_{\min}$ according to equation (3.56). The default values of ΔR_l are set according to [38] at 100 KeV as follows :

Implanted Species	ΔR_l [μm]
Boron	0.1267
Phosphorus	0.0393
Arsenic	0.0146
Antimony	0.0097

Table (6.6) : Default values for ΔR_l.

According to the above discussion, *dop* is implemented as follows :

```
function dop ( region_name : @char; $d_x$: real; $d_y$: real; $d_a$ :real)real;
var model : @char ; /* Distribution model */
    I_E: real; /* Implantation Energy */
    I_S: real; /* Implanted Species */
    I_D: real; /* Implanted Dose */
    RP: real; /* Projected range */
    SP: real; /* Vertical Standard Deviation */
    SPL: real; /*Lateral Standard Deviation */
    GAMMA: real ; /* Skewness for Pearson IV*/
    BETA: real ; /* Kurtosis for Pearson IV*/
begin
    model = get_ppar (region_name, model_id);
    I_S = get_ppar (region_name, is_id);
    I_E = get_option (region_name, energy_id);
    if I_E = 0;
    then I_E = default_energy;
    RP = get_option (region_name, rp_id);
    if RP = 0;
    then RP = proj_dev (0, I_E, I_S);
    SP = get_option (region_name, sp_id)
    if SP = 0;
    then SP = proj_dev (1, I_E, I_S);
    SPL = get_option (region_name, spl_id);
    if SPL = 0;
    then SPL = proj_dev (2, I_E, I_S);
    case model of:
        1D_GAUSSIAN : dop = oned_gauss (RP, SP, Dose, $d_x$);
        PEARSON_IV :
            GAMMA = get_option (region_name, gamma_id)
            BETA = get_option (region_name, beta_id)
            dop = pear_4 (GAMMA, BETA, RP, SP, Dose, $d_x$);
        2D_GAUSS :
    SPL = get_options (region_name, spl_id);
    if SPL = 0;
    then SPL = default_spl;
    dop = (RP, SP, SPL, dose, $d_x$, $d_y$, $d_a$);
end
```

The following figures are results of *dop* for boron in silicon at various implantation energies. Such profiles are illustrated using the graphics package developed for Albertina. The axes are not scaled as they are user defined, according to the discussion in section (6.7.1).

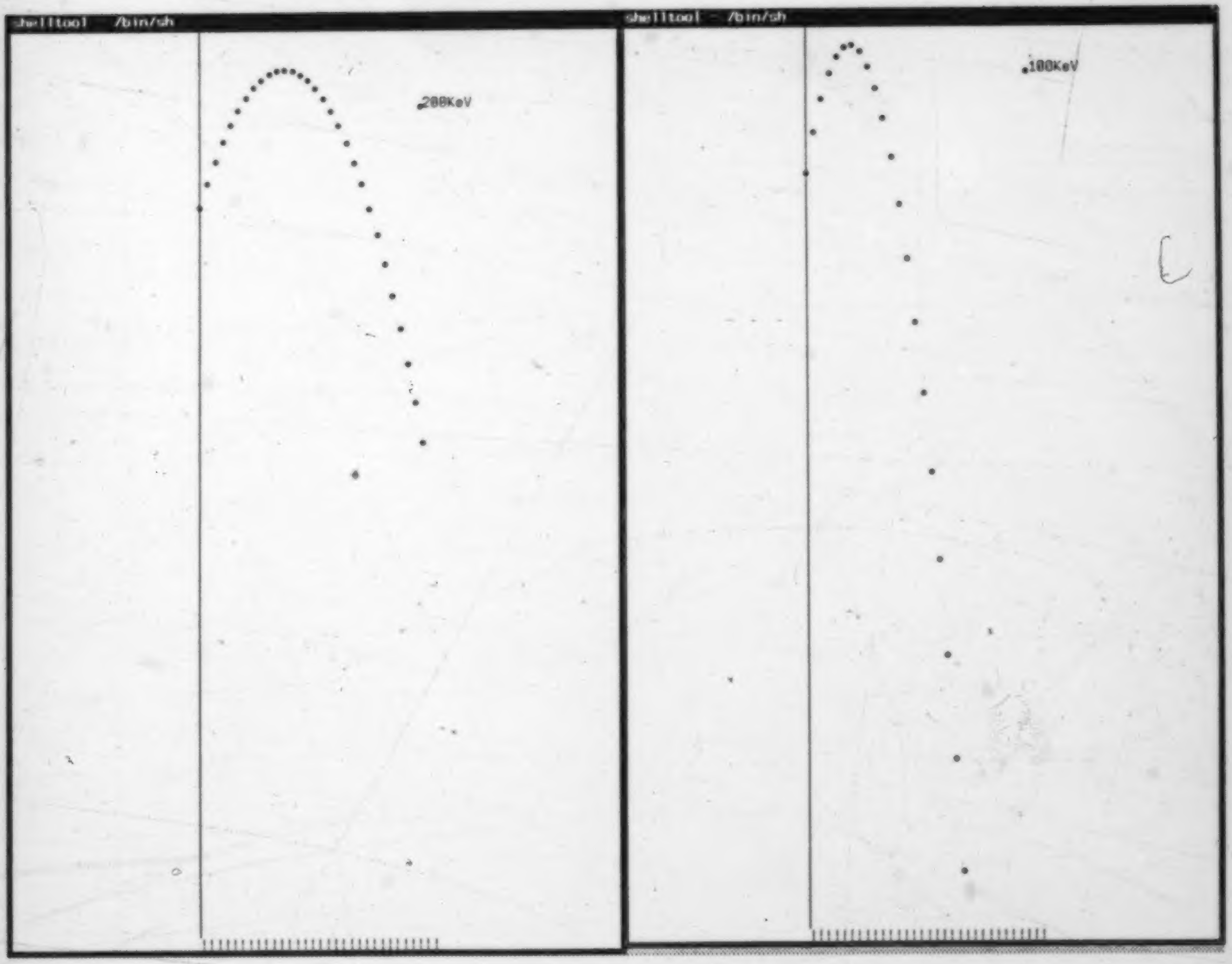

shelltool /bin/sh
200KeV
shelltool - /bin/sh
100KeV

Figure (6.9) : Plot of 1_D Gaussian doping profile of B in Si at 100 & 200KeV.

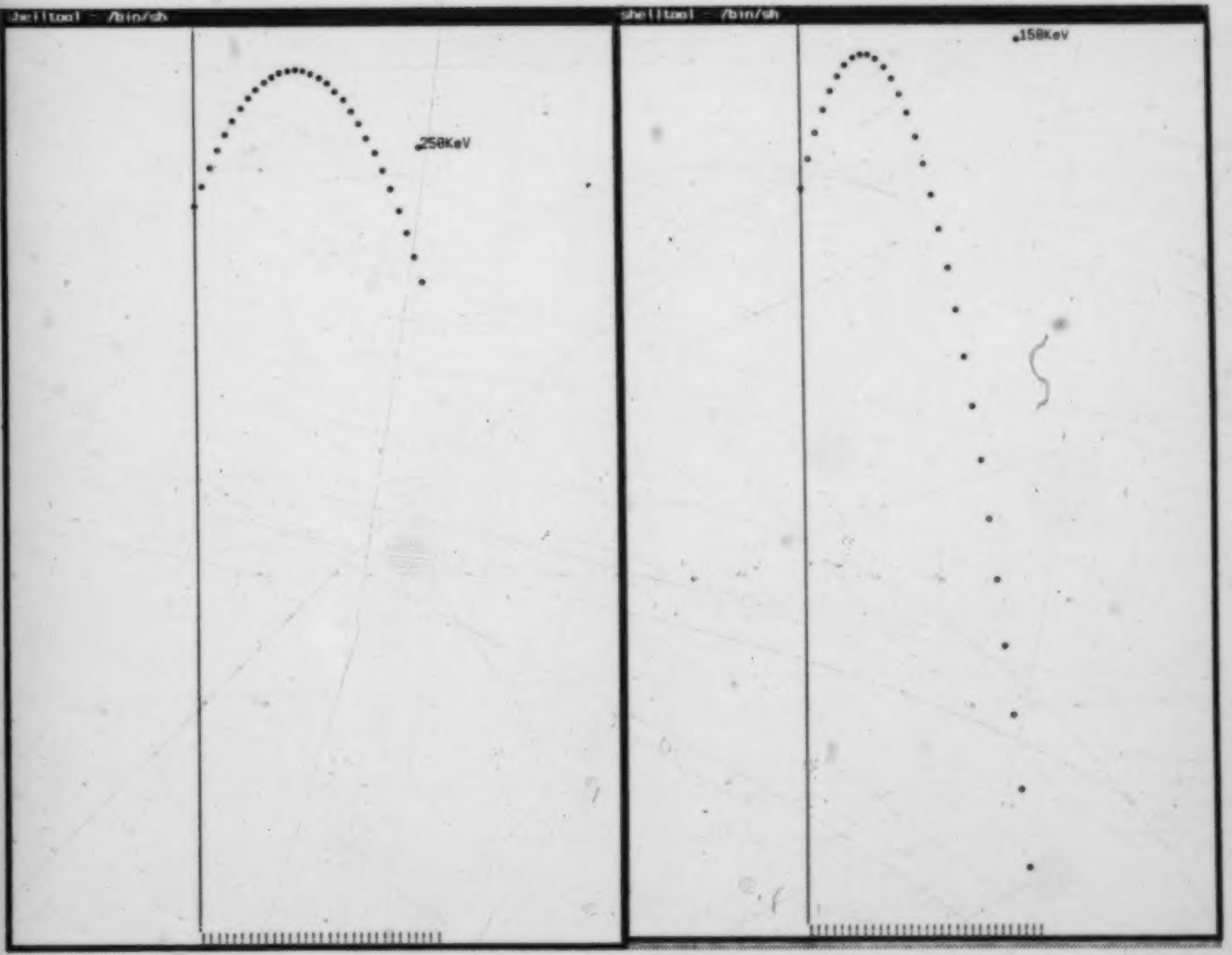


Figure (6.10) : Plot of 1_D Gaussian doping profile of B in Si at 150 & 250KeV.

6.5.4. Generation and Recombination PPSC (genere) :

The models offered to this PPSC, henceforth called *genere*, are SRH, Auger, Impact Ionization or a SUM of all three. Therefore the caller of *genere* must provide arguments according to the following table :

Model	n	p	ΔE_g	N_i	J_n	J_p	E
Auger	Yes	Yes	Yes	No	No	No	No
SRH	Yes	Yes	Yes	Yes	No	No	No
Impact Ion.	No	No	No	No	Yes	Yes	Yes
SUM	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Table (6.7) : Arguments of *genere* PPSC.

In this PPSC, the calculation of *genere* is straightforward, it is calculated for all three in case of SUM as per equation (3.98). The temperature dependence of equations (3.94-3.97) inclusive is calculated according to the extracted temperature from the physical descriptor node.

6.5.5. The Expansion of OPTIONS :

During our literature survey of models, parameters and their variables, we noted a large range of empirical and experimental values for each variable. Accordingly *OPTIONS* was expanded to cover other parameters. Hence, the procedure *get_options* was modified and incorporated with the relevant PPSC operations, further increasing the system flexibility. The current implementation of OPTIONS allows the user to define the following variables :

PPSC	Model (eqn. #)	Optional Variables
genere	S_R_H (3.86)	$\tau_n, \tau_p, N_n^{ref}, N_p^{ref}$
genere	Auger (3.90)	C_n, C_p
band	Thurmond (3.105)	$E_g(0), \alpha_t, \beta_t$

Table (6.8) : Expansion of OPTIONS.

OPTIONS "Action module" is invoked in *tty_subwindow*. Once invoked, the user is prompted to enter the values of each variable.

6.6. User Interface vs. External Modules :

The relationship between the user interface and the external modules is mainly two mechanisms summarized as follows :

1. Invoking : Upon the user's command, this mechanism allows the user interface to invoke the external module. The module is a separate program not linked to the main core of the user interface.
2. Linkage : On invoking a module, it is the responsibility of the user interface to insure the availability of the appropriate data required by the module. This data should also be in a format compatible to that used by the module.

In the following subsections we shall discuss these two points and review certain previous definitions to accommodate the details of these mechanisms.

6.6.1. Invokement of External Modules :

In section (3.2), we have set "Concurrency" as one of the objectives considered in building the Albertina User Interface, especially for time consuming operations. This means that any of these modules will be executed from within the main core of the user interface. This is accomplished with two steps. First using the Unix system call *fork* [59, 60], which creates two identical copies of a process. The first process is denoted as the *parent* the second is denoted as the *child*. All of the parent's information are inherited by the child, yet the forked process (child) has its own data and stack segments. The *forked* process could be used with *wait* system call, allowing the execution of the parent process to be suspended until the termination of the child process. In our implementation this aspect is redundant, as each of the external modules is an independent interactive process on its own that is fundamentally terminated upon the user's request. Furthermore, the suspension of the parent process in our case will defeat the idea of concurrency. The second step of invoking the external module is using the *exec* system call, which overlays the running process with a new program starting at its entry point. We have to note that there is no return to the calling program after a successful *exec* and its core is lost.

In Albertina we applied these calls through a module, henceforth called *alba_inv*, where the user interface will be *forked* into a parent and child. The parent remains active and accessible to the user at any point in time as it does not wait for the child to terminate. On the other hand, the child process will use the system call *exec*, as a result the external module will overlay the child process. The external module will run until terminated by the user. Meanwhile, the core of the child process is lost and the process will cease to exist. The merit of this mechanism is that it does not result in unnecessary CPU time consumption or memory occupation.

Both the mesh generator and the PDE solver are invoked by this mechanism. Therefore, in the following subsections we will separately discuss the linkage of each of these packages.

6.6.2. Mesh Linkage :

The Mesh task is to discretize the device in the following phases :

1. Preliminary discretization : Each region will be geometrically divided into a certain number of elements, according to a user given criteria.

2. Refinement : Further discretization is performed in order to maintain a certain value of the doping gradient between any two nodes in the mesh.

The details of these discretization schemes are beyond the scope of this work, but the task of providing the necessary data for meeting the above criteria is one of the goals of this work. The data provided to the mesh generator by the user interface will accommodate these two discretization stages as follows :

6.6.2.1. Geometry Segmentation :

In chapter 5, we have discussed the method by which the user can define any of the geometric forms provided by Albertina. The final result is basically a tree presentation of the algebraic equation describing the geometry. For example, in the case of a circle the equation is :

$$(x - xcr)^2 + (y - ycr) = rd^2 \tag{6.1}$$

where *xcr*, *ycr* and *rd* are the values of the nodes of which the geometric sub-tree is constructed. The "Mesh Generator" operates on the geometry of each region in a piecewise manner, i.e. each region has to be redescribed as a polygon. Hence, a transformation of the region description from a continuous contour to a segmented description is required. This transformation is obviously required only for two of the shapes provided by the system, the first is the circle, the second is the ellipse.

The geometry segmentation is implemented by reviewing the algebraic equations of the geometric forms. The equations are rewritten in the form of "Parametric Equations". This segmentation is illustrated in figure (6.11) and is implemented as follows :

1. Circle : Has the following parametric equation :

$$(x - xcr) = r \cos \beta \tag{6.2}$$
$$(y - ycr) = r \sin \beta \,. \tag{6.3}$$

2. Ellipse : This geometric form is described by the following parametric equations :

$$x = majx \cos\beta \tag{6.4}$$

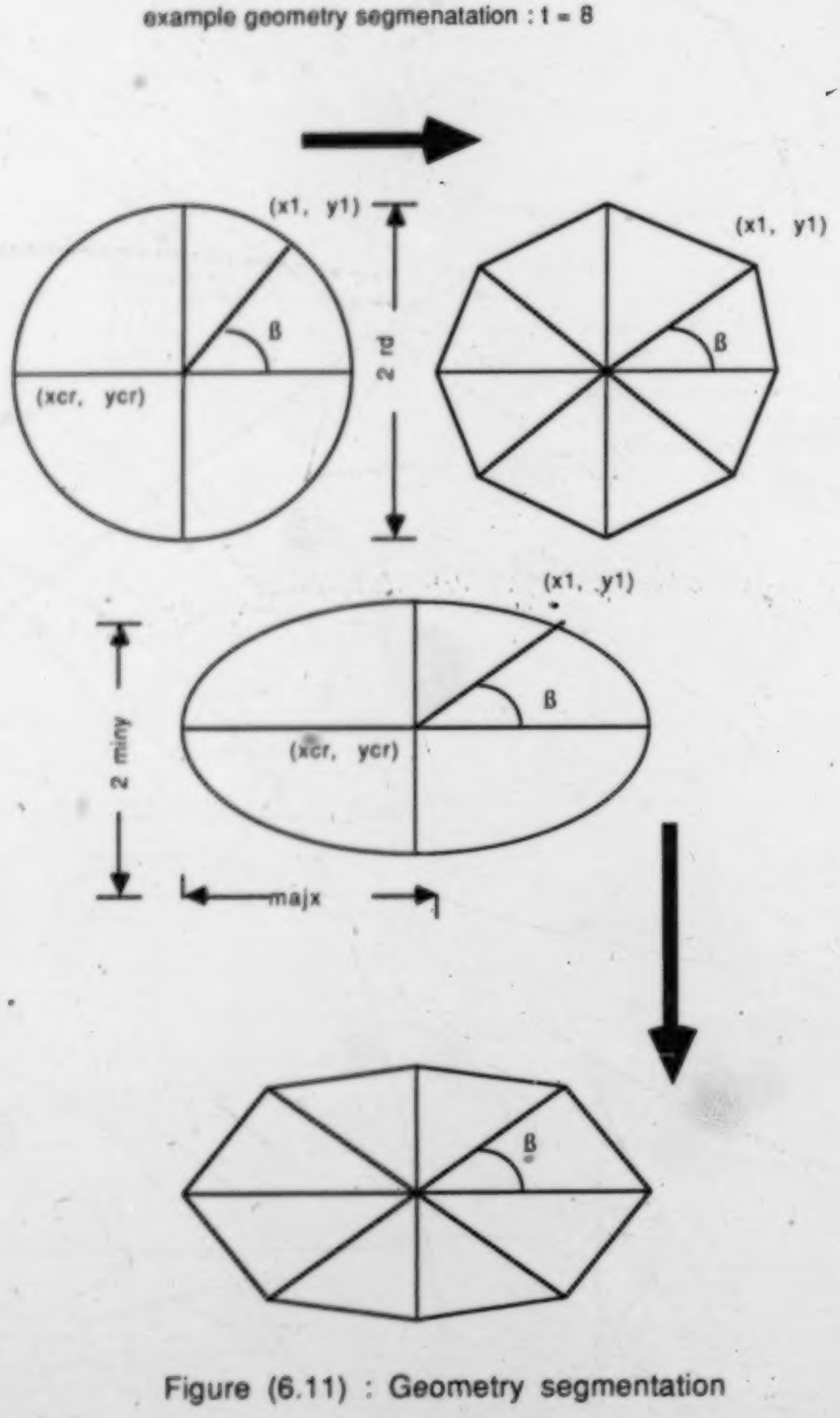

example geometry segmenatation : t = 8
(x1, y1)
ß
2 rd
(xcr, ycr)
(x1, y1)
ß
(x1, y1)
ß
2 miny
(xcr, ycr)
majx
ß

Figure (6.11) : Geometry segmentation

$$y = miny \sin \beta \tag{6.5}$$

Considering these equations, the user is prompted to enter the number of triangles **t** to be generated in the region. The linear distance between any two nodes on the circumference **d** should be calculated according to [71] :

$$d = \sqrt{\frac{2A}{t}}, \tag{6.6}$$

where A is the area of the region. Therefore, we generate an array of points redescribing the region in terms of segments as follows:

point[0] = (xcr + ψ, ycr)

point[n] = (x_parametric_function (nθ), y_parametric_function(nθ))

n = 1, 2, ..., d-1

where ψ = xcr for a circle,

and ψ = majx in the case of an ellipse. The angle θ is calculated as :

$$\theta = \frac{360}{d}.$$

In this manner, we start at the right most point of the geometric form and proceed in an anti-clockwise direction, as required by the mesh generator. The output by this procedure is incorporated with another data file describing the other elements and handed over to the mesh generator for further processing and discretization. Although region segmentation is basically a user interface function, for practical reasons, it should be relocated as a part of the mesh generator. This decision has been made because such a function related more to the mesh than to the user interface.

6.6.2.2. Refinement According to the Doping Gradient :

The concern of this work with respect to this criterion of refinement is to provide the doping concentration at any point (mesh node) within a region in a two dimensional Cartesian plane. The doping gradient between any two nodes is considered as the difference between of the doping concentration at each node. In section (6.5.3), we have discussed the *dop* PPSC which will provide the doping concentration at each node. Therefore a copy *alba_pipe* is placed within the mesh generator to allow *piping* to the PPSC function as previously discussed. The mesh will call *dop* for each node and subtract the returned values in case the gradient between them is required.

6.6.3. PDE Solver Linkage :

The PDE solver is invoked through *alba_inv* and has access to all the PPSC library by adding to it a copy of *alba_pipe*. The PPSC library provides the PDE solver with the values of carrier mobility μ, bandgap narrowing ΔE_G, and Generation and Recombination *(R - G)*. The doping concentration of the discretized device, i.e. at each node in the grid, is passed over from the mesh generator as it has already been acquired for refinement. The PDE solver will require one more parameter which is the boundary condition. This parameter could be extracted from the data pool by adding a copy of *get_ppar* in the PDE code.

6.6.4. The Mesh and PDE Solver Sequence :

In figure (6.12), we illustrate the system within the above discussion and incorporate our implementation of the previous chapters. In this figure invokable modules and operations are displayed in boxes. The data pools are displayed within circles. Hence, the figure is discussed as follows:

The user communicates with the system through the transparent graphical layer through which all modules are invoked. The modules are accessed either through direct panels or through menus. These modules are either :

a. Description Tools : Divided into Geometric and Physical tools. They virtually correspond to the *Shape, Ishape and PPar* UPOs. The description data then will reside in their respective data pools.

b. External Modules : Namely the Mesh Generator and the PDE Solver, are invoked and monitored by the user interface. The modules utilize their

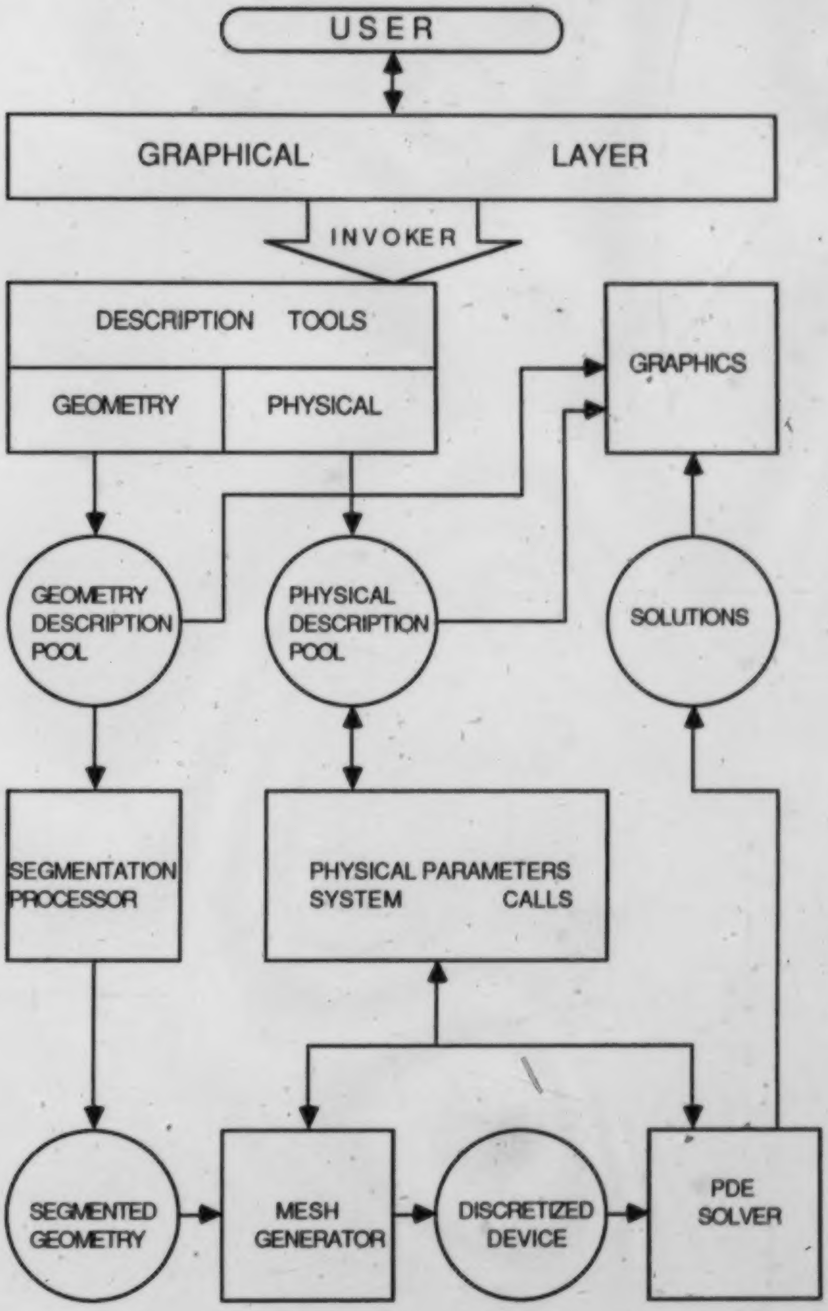

USER
GRAPHICAL LAYER
INVOKER
DESCRIPTION TOOLS
GEOMETRY
PHYSICAL
GRAPHICS
GEOMETRY DESCRIPTION POOL
PHYSICAL DESCRIPTION POOL
SOLUTIONS
SEGMENTATION PROCESSOR
PHYSICAL PARAMETERS SYSTEM CALLS
SEGMENTED GEOMETRY
MESH GENERATOR
DISCRETIZED DEVICE
PDE SOLVER

Figure (6.12) : Integration of Albertina relevant data pools. Furthermore, through pipes the modules access any of the PPSC library functions. The final result resides in the "Solutions" pool.

c. Auxiliary Functions ; Represented in this figure is the graphics package. It is also invoked through the interface, and according to a user command will graphically display a certain view of the device.

6.6.5, Monitoring and Reliability :

In section (2.5) as we discussed Albertina's requirements we set data monitoring as one of the user interface tasks. In figure (6.12), we note that the data required by the mesh are the "Geometry Description Pool". The mesh and the PDE solver use the PPSC which needs the "Physical Description Data Pool". When a user requests the mesh to be invoked, the system will run a completion module to insure that all the geometric and physical parameters have been defined. If an undefined parameter is encountered the user is notified and mesh invokement is aborted. After this check is performed the "Segmentation Processer" is automatically invoked from within the mesh generator, thus guaranteeing the existence of the "Segmented Geometry" data. On the other hand, the PDE solver requires the "Discretized Device" data. In this case, the user interface will monitor this pool prior to invoking the PDE solver. Monitoring is performed in order to :

a. Insure the existence of the pool, if the file containing the discretized device does not exist the invokement is aborted and the user is notified.

b. The system will check the time t_{dd} at which this pool has been created and is compared to the time t_{sg} at which the segmented geometry pool was last modified. If

$$t_{dd} < t_{sg} \tag{6.7}$$

then the "Segmented Geometry Pool" has not been discretized, i.e. although a "Discretized Device Pool" exists it is not the result of the latest update of either physical or geometric data pool. In this case, the process of invoking the solver is aborted and the user is advised.

All error messages are self explanatory and are displayed in the *text_subwindow*. Another way of assisting the user is an on-line *HELP* facility that is provided as a panel and is classified as an "Auxiliary Function". This operation will provide a quick review of any menu selectable system operations. The help information is displayed on the screen in the *text_subwindow*. The scrolling facility of this subwindow is used to allow the user to review the information.

6.7. Graphics Package :

Reviewing figures (4.1) and (6.12), we identify a third "External Module" that has been included in the new structure of Albertina, namely the "Graphics Package". As previously mentioned the user in some cases can best analyze and inspect data once they are graphically displayed. Hence, the importance of incorporating such a package.

The data requiring display are basically the device geometry, the discretized device (the mesh), results from the PDE solver, and the distribution of certain physical parameters such as the doping profile. We can therefore classify the required graphics package to be composed of :

1. Two dimensional display routines : Capable of displaying any two dimensional information such as the geometry, the mesh and the 2_D distribution of the physical parameters.
2. Three dimensional display routines : Certain variables are displayed in three dimensions. These routines are responsible for such displays. The variables are mainly the results of the PDE solver, such as ϕ the electrostatic potential distribution.

While developing this work we have considered developing both classes of routines. However, due to the maturity of the PDE solver, the main user of 3_D routines, its adaptability to these routines would have required a great deal of effort and time on the part of the PDE solver developers; hence, we have concentrated on two dimensional displays only.

The first issue we have confronted during the development of the graphics routines was : *Where on the screen would the output be displayed ?* Reviewing figure (6.1) the options were limited to either the *text_subwindow* or *tty_subwindow*. The disadvantages of using these subwindows could be explained in the following points :

1. Occupation : Creating a display in either of these subwindows would prohibit the user from using the subwindow's main function. As a result the system's availability will be reduced.
2. Multiplicity : Using a subwindow would also limit the user to only one display at any time, where as the user might require multiple displays of different facets of the device.
3. Retainability : Defined as the ability to retain the display of a subwindow after an overlapping subwindow has been removed. For example, if subwindow **B** has been created overlapping an existing window **A,** the system should be able to refresh the display in window **A** once **B** has been removed. This feature is required as a result of solving the multiplicity problem.
4. Closure : A window in *Sunview* is capable of being opened to display its data or closed to an *iconic* form. In this form the window waits for the user to reopen it and acquire either its data or function.
5. Display Dimensions : The *text* and *tty* subwindows are confined to two dimensional displays. Therefore, using them would limit the future expansion of the package to incorporate 3D routines.
6. Invokement : Running the graphics routines through these subwindows would have required a mechanism other than *alba_inv* to invoke it. This is because the graphics in this case will be a part of the interface rather than being a separate program.

Realizing all of these features within the *Sunview* definition of the subwindows is cumbersome for some and not feasible for others. We therefore, resorted to developing a separate graphics tool in the form of an invokable subwindow, henceforth, called *alba_gfx*. This sub-window attains all of the above features with a moderate consumption of CPU time. The number of displays is only limited by the number of children allowed per process by the Unix operating system. The two dimensional routines are run within this sub-window and are invoked by *alba_inv* as follows :

alba_gfx < gfx_routine > < module_arguments >.

Our implementation of the two dimensional routines are discussed in the following sub-section. The routines utilize SUNCGI [66] and are run by placing the routine name as the *gfx_routine* argument. They are used to display the device geometry and the distribution of physical parameters. Finally, our routines were not incorporated with the mesh generator due to its independent graphical interactive nature.

6.7.1. Plotting Routine :

This routine named *alba_plot* is invoked with the following arguments :

alba_plot <file_name> < x_max> <y_max>,

where *file_name* is the name of the file containing the data to be displayed. Each file record, composed of two fields the x and y representing the coordinates of a point, is read and displayed. The second and third arguments *x_max* and *y_max* are the maximum coordinates on the x and y axis respectively, these arguments are used by *alba_gfx* to create an appropriately sized window. *Alba_plot* is used to display the results of the PPSC library in chapter 3.

6.7.2. Two Dimensional Geometric Display :

In another implementation of SUNCGI, the geometry pool is displayed in an *alba_gfx* window. This routine, henceforth called *display*, is invoked with one argument as follows :

display < file_name >,

where *file_name* is the name of a file containing the geometry description of the device. It is created by the *format_conversion* procedure that is used by the *Save, List* and *Load* UPOs. Thus the file is of the format shown in table (5.1). The procedure used to create such a file is similar to that used by the UPOs with the following differences :

1. The file in which the data are stored is created by this procedure.
2. The file will contain only the geometric information. Thus, the root argument of *v_browse* should be the root of the geometry sub-tree.

This procedure will prepare the geometry description of a region for *display* in a file named "GEOMETRY" as follows :

```
procedure prepare_geometry_description (region_name : @char)
var rg_ptr : region_pointer ;
root : geo_pointer :
geometry_file : file_ponter;
    destination : @function_pointer ;
begin
    geometry_file = @rewrite (GEOMETRY);
    /* Create the file GEOMETRY, assign the pointer
    geometry_file to it*/
    rg_ptr = locate (region_name, 0, NULL);
    /* Use the locate SPO to find the region*/
    destination = @format_converter (geometry_file);
    /* Direct the file pointer to the format conversion procedure,
    set the destination before vertical browsing*/
    root = rg_ptr.g_link;
```

```
        /* Find the root of the geometry sub-tree*/
        v_browse (destination, root);
    end
```

The routine *display* use's cgi primitives that will draw a variety of geometric shapes. It is implemented according to the following algorithm :

```
    procedure display (f_ptr : file_pointer)
var node_id : integer;
    att1: @char;
    att2: integer;
begin
    initialize_cgi_graphics ();
    set_window();
    create_user_grid();
    verify_file_format();
    while not eof
        get_all_fields( f_ptr, node_id, att1_att2);
        if node_id = 20
        then case att1 of
            alba_shape : draw_shape (f_ptr);
        else
            get_all_fields( f_ptr, node_id, att1_att2);
        end
end
procedure draw_shape (f_ptr) var shape_par_1 :integer;
    shape_par_2 :integer;
    shape_par_3 :integer;
begin
```

```
while node_id != 20 ;
    get_all_fields( f_ptr, node_id, att1_att2);
    case att1 of
        shape_attribute : shape_par_x = att2;
    cgi_shape_output_primitive (shape_par1, shape_par_2,.....);
end
```

Display is used to illustrate the geometric forms in the following example.

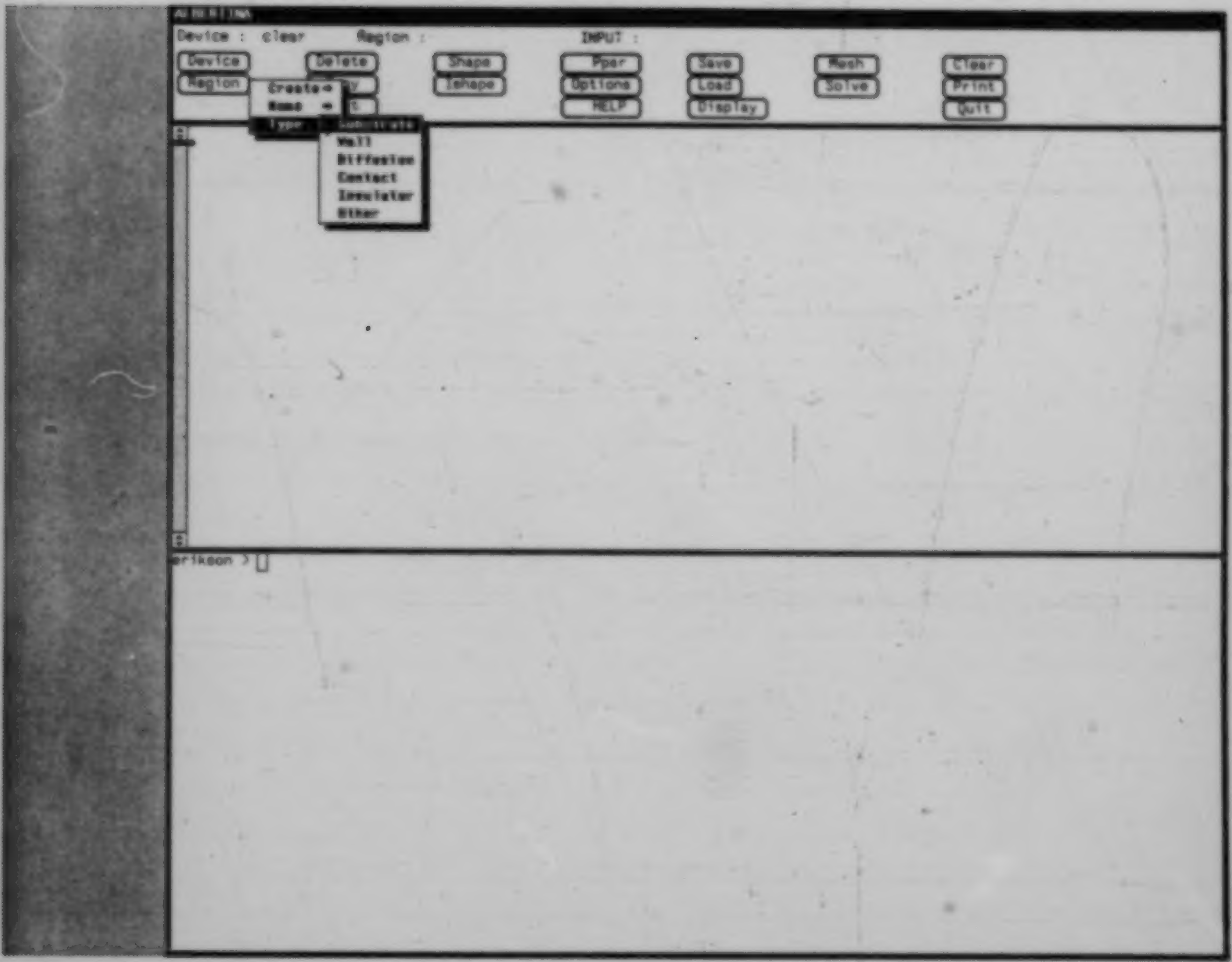
6.8 Example of a Typical Simulation Session :

We would like to end this text with an example of using the user interface session. In the following pages we present several *screendumps*, i.e. the computer terminal screen as seen by a user during a simulation session. The screendumps represent parts of a session to simulate a semiconductor device.

The simulated device in this session is a Hall plate. The device is a silicon substrate with two ohmic contacts on two opposite sides. As discussed in subsection (3.1.4), the presence of the semiconductor device in a magnetic field will subject the charge carriers to a deflective force. In the case of a Hall plate, the deflection of the carriers will create an imbalance between the voltages appearing at each ohmic contact. The difference between these voltages is proportional to the applied magnetic filed induction vector $\vec{B}$. Thus, the Hall plate could be used to sense magnetic fields.

In the following pages we present the steps done to describe the geometry and assign the physical parameters of this device. These steps are implemented by selecting the appropriate panel and invoking either a menu hierarchy or a certain function. For clarity purposes, menus are displayed in separate figures on data free screens.

Figures (6.13) and (6.14) show the menus used to create a region and to define the device as a whole. The selection of the *Region* panel in figure (6.13) invokes two menus. The first menu has three operations. If *create* is selected a new region is created, the name of the region is displayed and the user is advised of its creation. If the user chooses to change the region's name, the operation *Name* is selected. The user should provide the new name in the string panel *INPUT*. The third operation of the menu displayed in figure (6.13) is *type*. If this operation is selected a new menu appears allowing the user to define the region's type. In this case the created region is named *one* and is a *substrate*. When the first region is created, the system will automatically create the device cluster of figure (5.4). The user can then define the name and type of the device. This is done by the menus in figure (6.14). These menus pertain to the *Device* panel. Similar to naming a region, the device is named by using the *INPUT* string panel. Selecting the *type* operation will invoke another menu allowing the user to define the device type. In this example the device is named *Hall_plate* and is of type *Bipolar*. The user interface will then erase the device name form the INPUT panel and display it in the *Device* string panel. The results of these two operations can be seen in figure (6.15). In this figure, the *List* panel has been invoked, the operation pertaining to it will take the character string in the *Region* string panel as an argument. This argument is used to locate a region whose name corresponds to it. In figure (6.15), the wild card (*) is entered, thus the whole device is displayed. The *listing* of the device is shown in the text_subwindow, where the name and type of the region and the device are shown. The shape and physical parameters of region *one* are denoted as *undefined*. Defining the region geometry requires using the menu in figure (6.16). This menu is invoked by selecting the *Shape* panel, with the condition of having the region's name in the *Region* string panel. The menu allows the user to define the region shape, in our case the device is a polygon. Invoking *Ishape* will allow the user to instantiate the geometry sub-tree. The user is prompted to enter the values of the device geometric parameters in the text_subwindow, this is shown in figure (6.17). The values in the text_subwindow indicate the name and type of the region, followed by the type and value of the root node of the geometry sub-tree, *type = shape, value = polygon*. The rest of the sub_tree is shown as the type and value of each node, *y_pol* is the y coordinate of the polygon vertex, *x_pol* is its x coordinate. The lisiting of this region indicates that the physical parameters are not yet defined. The physical definition of the region is implemented by invoking the *Ppar* panel, which will invoke a set of menus as shown in figure (6.18). The first menu in this figure shows the physical parameters, which are the attributes of the physical descriptor node. Selecting an attribute will invoke another menu which contains the models of this physical parameter. Figure (6.18) shows the *Ppar* menu with the physical parameters and the mobility menu with the various mobility models. If the user would then like to display the device geometry the *Display* panel should be selected. This panel, shown in figure (6.19), allows the user to display the geometry, the mesh, doping profile, or the electrostatic distribution. Although the system in its current state cannot display these device dimensions, the menu has included them for future system expansion. In figure (6.18) the *Geometry* is selected from the *Display* menu, figure (6.20) shows the result of this selection. The device geometry is displayed in a newly created window *alba_gfx*. In this window, the graphics package is invoked and the geometry is displayed according to the discussion in section (6.7.2). The interface is still active as both are concurrently executed. In the interface text_subwindow, the geometry and physical description of the device are listed. The last figure (6.21) shows the screen after invoking the mesh generator. It is invoked by selecting the *Mesh* panel. The figure includes the geometry display and the discretized device, the interface graphical layer is visible in the background. Invoking the mesh generator is implemented according to the discussion in sections (6.6.1) and (6.6.2). In this case geometry segmentation was not required because the device is described as a polygon.

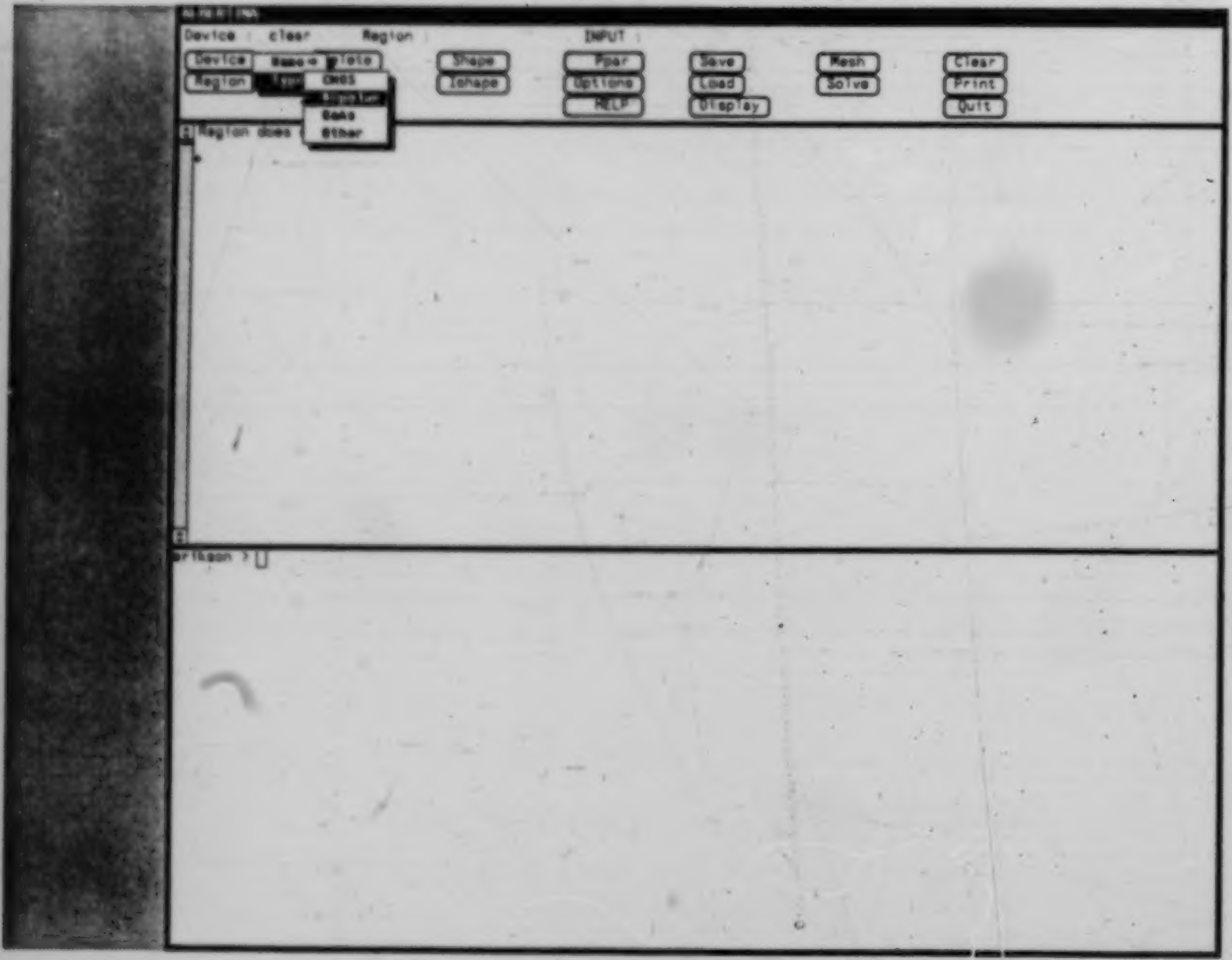


Device : clear Region : INPUT :

erikson >

ALBERTINA
Device
Region
Delete
Shape
Ishape
Pper
Options
HELP
Save
Load
Display
Mesh
Solve
Clear
Print
Quit
Create
Name
Type
Substrate
Well
Diffusion
Contact
Insulator
Other

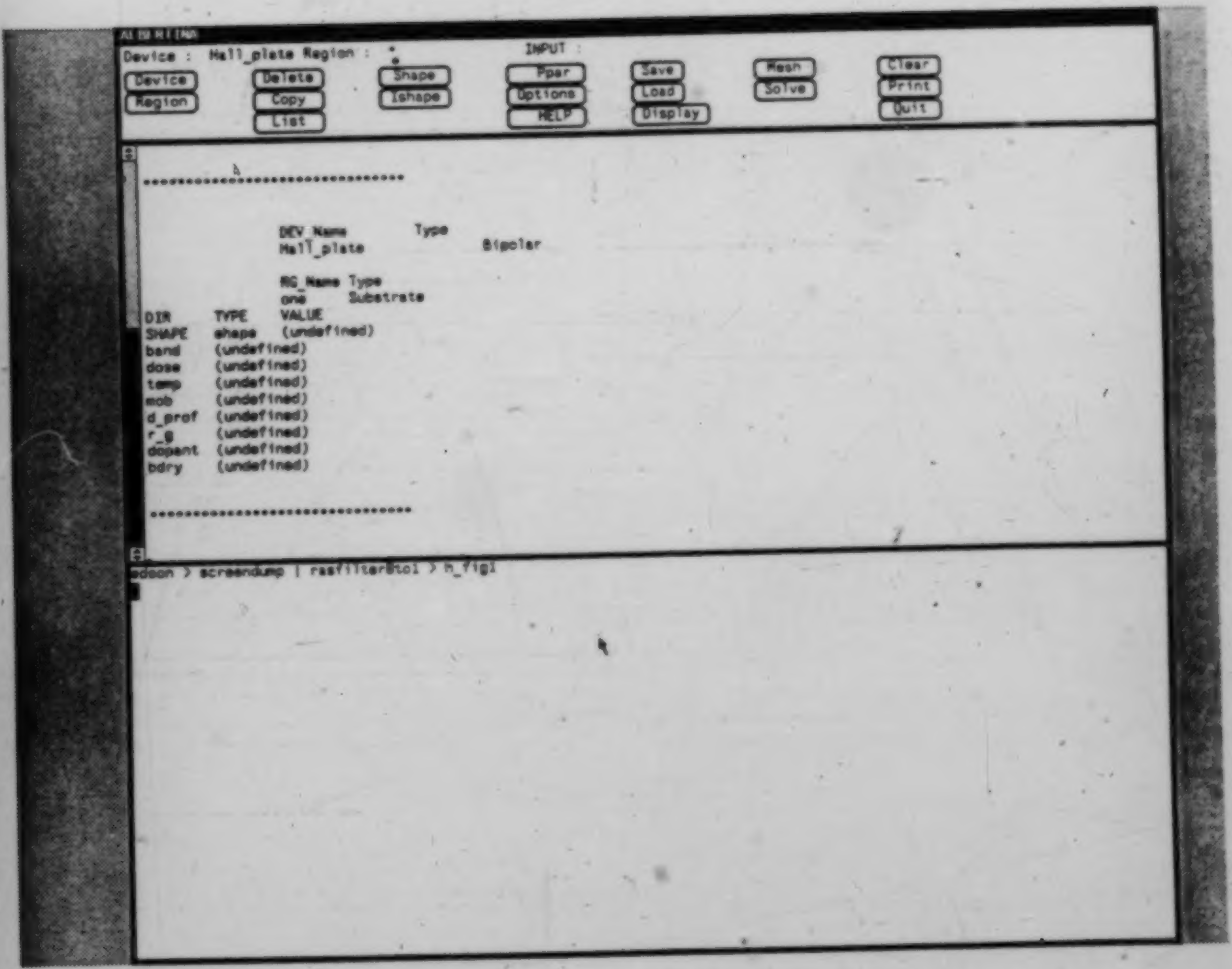
Figure (6.13) : Menus of Region Panel.



```
Device : clear      Region :            INPUT :
Region does
rikson >
```

Device
Region
Name ⇒
Type
CMOS
Bipolar
GaAs
Other
Shape
Ishape
Fpar
Options
HELP
Save
Load
Display
Mesh
Solve
Clear
Print
Quit

Figure (6.14) : Menus of Device Panel.

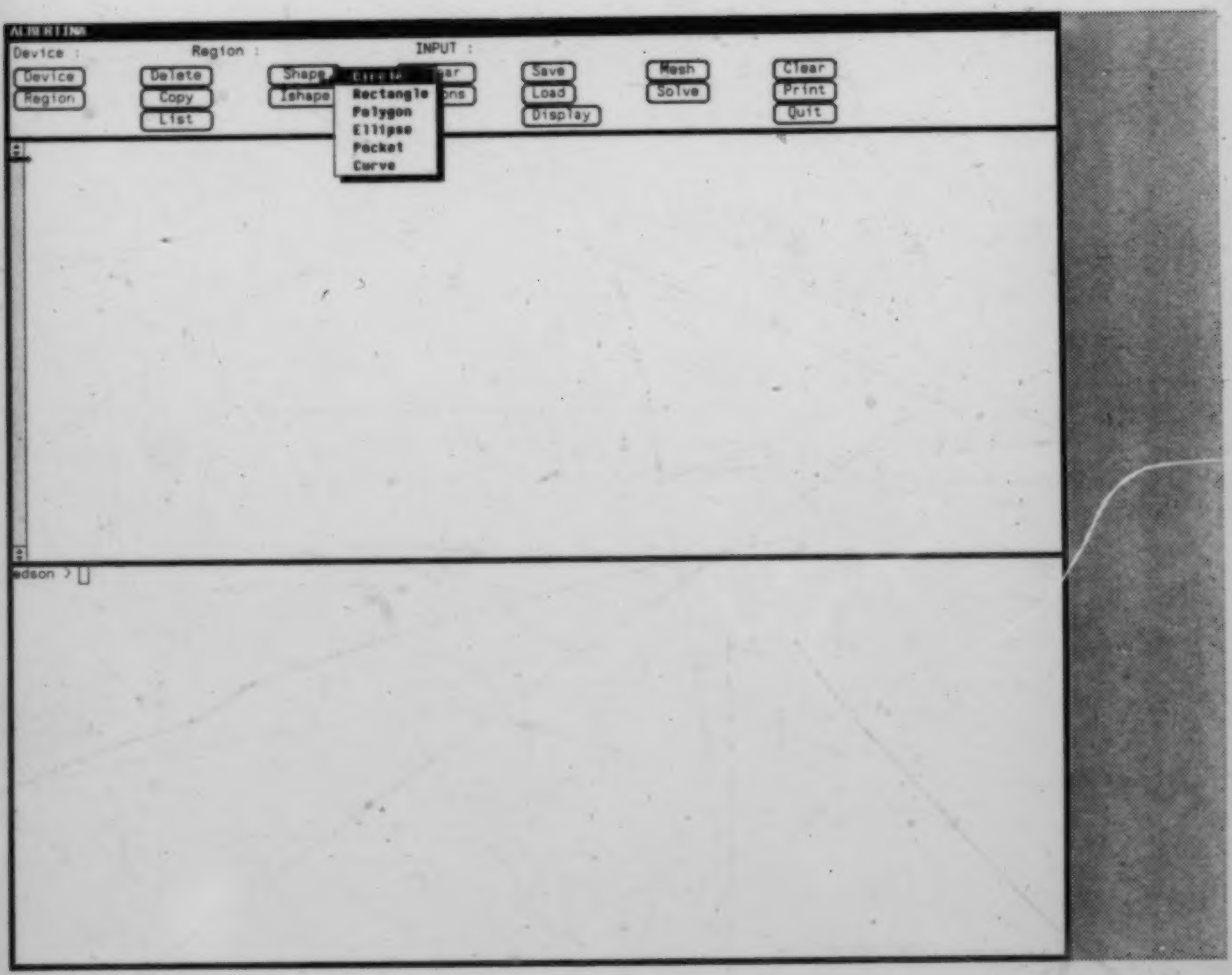

Device
Region
Delete
Copy
List
Shape
Ishape
Ppar
Options
HELP
Save
Load
Display
Mesh
Solve
Clear
Print
Quit

```
Device :  Hall_plate Region :  *                    INPUT :
```

```
******************************


                    DEV_Name         Type
                    Hall_plate                 Bipolar

                    RG_Name Type
                    one     Substrate
DIR      TYPE       VALUE
SHAPE    shape      (undefined)
band     (undefined)
dose     (undefined)
temp     (undefined)
mob      (undefined)
d_prof   (undefined)
r_g      (undefined)
dopent   (undefined)
bdry     (undefined)


******************************
```

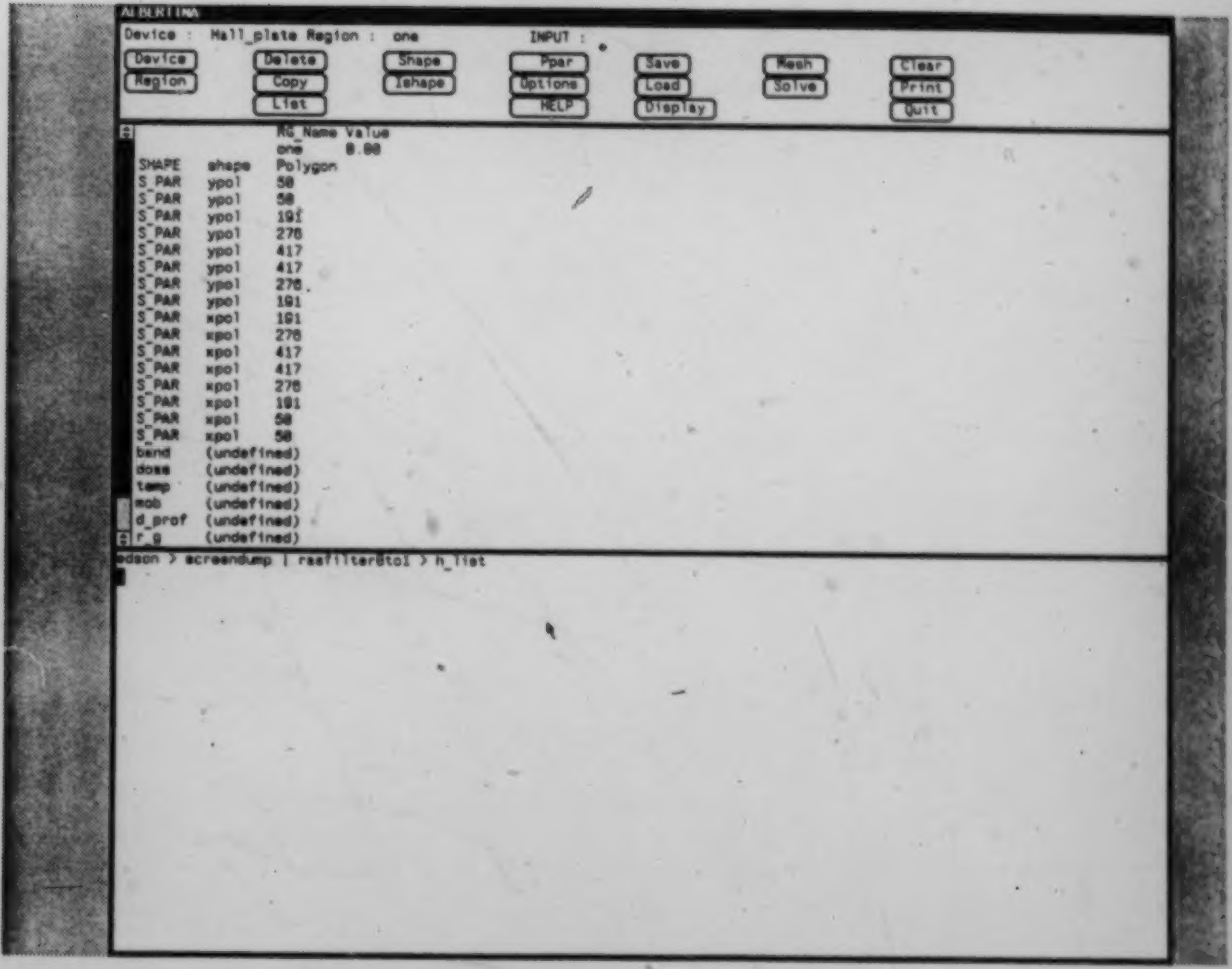
```
edoon > screendump | rasfilter8to1 > h_fig1
```

Figure (6.15) : Listing of a device.


Device :
Region :
INPUT :
Device
Region
Delete
Copy
List
Shape
Ishape
Circle
Rectangle
Polygon
Ellipse
Pocket
Curve
Save
Load
Display
Mesh
Solve
Clear
Print
Quit

```
edson >
```

Figure (6.16) : Menu of Shape Panel.

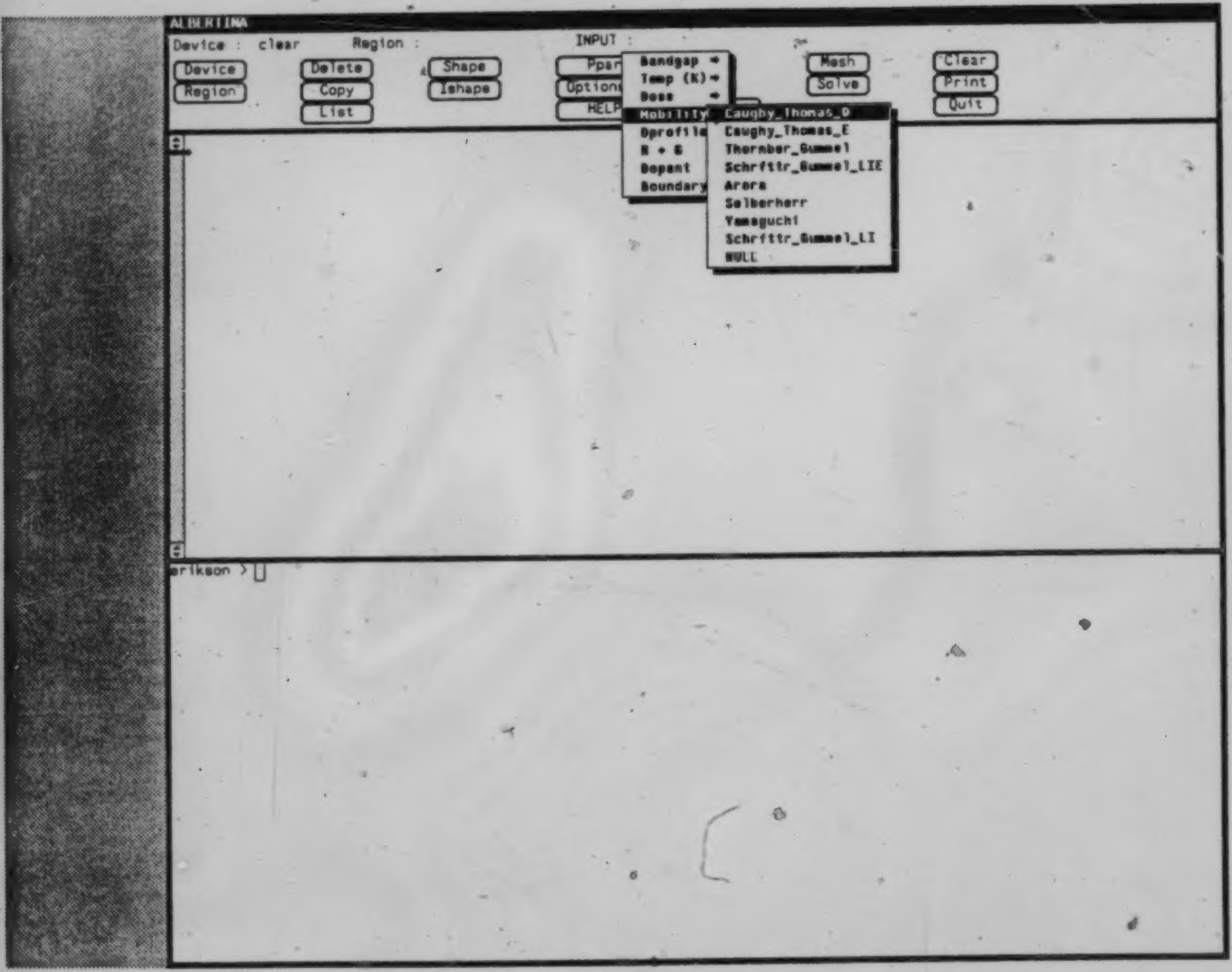

ALBERTINA
Device
Region
Delete
Copy
List
Shape
Ishape
Ppar
Options
HELP
Save
Load
Display
Mesh
Solve
Clear
Print
Quit

```
Device :  Hall_plate Region :  one          INPUT :
```

```
                    RG_Name Value
                    one     0.00
SHAPE     shape     Polygon
S_PAR     ypol      50
S_PAR     ypol      50
S_PAR     ypol      101
S_PAR     ypol      270
S_PAR     ypol      417
S_PAR     ypol      417
S_PAR     ypol      270
S_PAR     ypol      101
S_PAR     xpol      101
S_PAR     xpol      270
S_PAR     xpol      417
S_PAR     xpol      417
S_PAR     xpol      270
S_PAR     xpol      101
S_PAR     xpol      50
S_PAR     xpol      50
bend      (undefined)
dose      (undefined)
temp      (undefined)
mob       (undefined)
d_prof    (undefined)
r_g       (undefined)
```

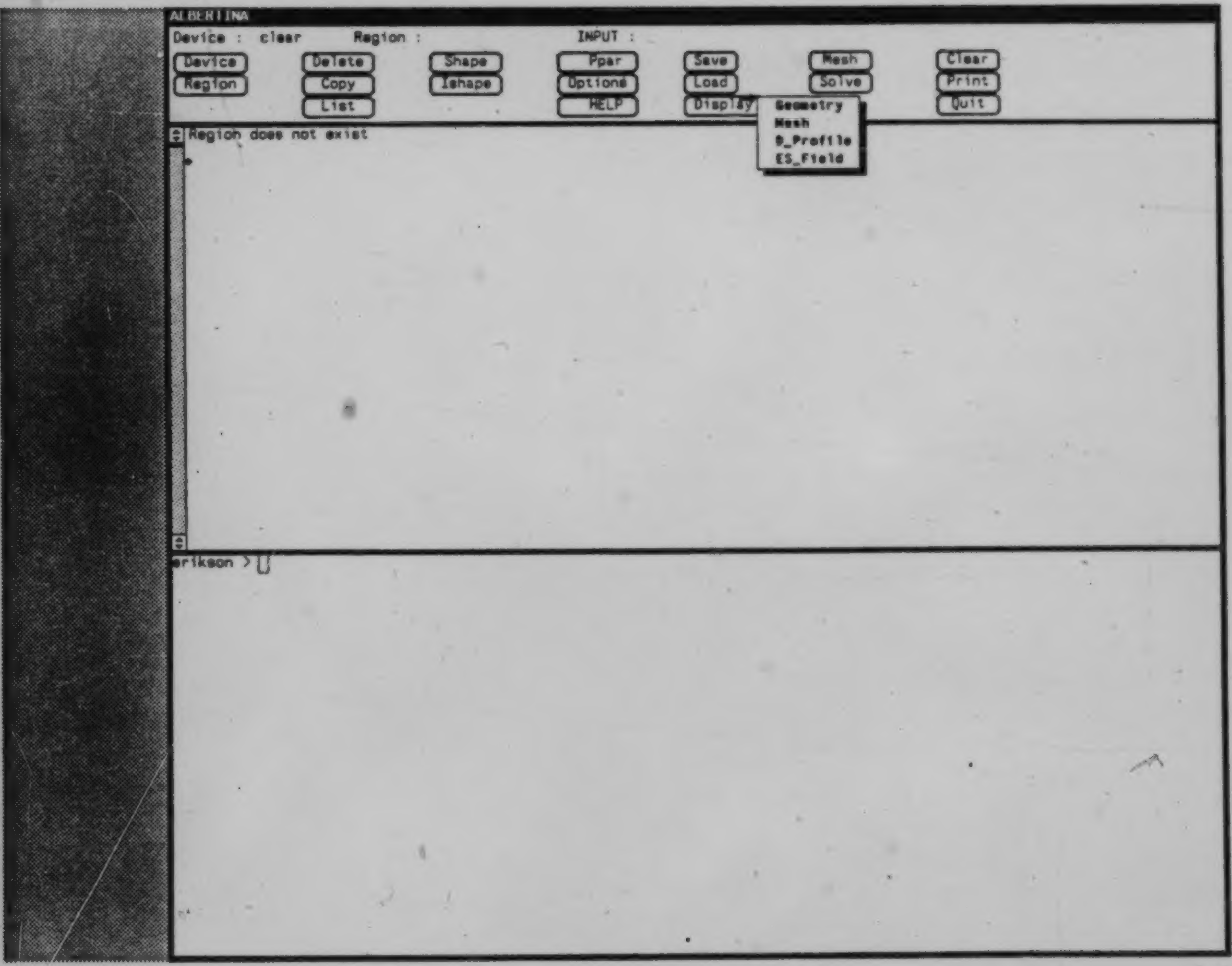
```
edson > screendump | rasfilter8to1 > h_list
```

Figure (6.17) : Listing of device geometry.


ALBERTINA
Device : clear
Region :
INPUT :
Device
Region
Delete
Copy
List
Shape
Ishape
Ppar
Options
HELP
Bandgap ➡
Temp (K) ➡
Dose ➡
Mobility
Dprofile
R + G
Dopant
Boundary
Caughy_Thomas_D
Caughy_Thomas_E
Thornber_Gummel
Schrfttr_Gummel_LIE
Arora
Selberherr
Yamaguchi
Schrfttr_Gummel_LI
NULL
Mesh
Solve
Clear
Print
Quit

```
erikson >
```

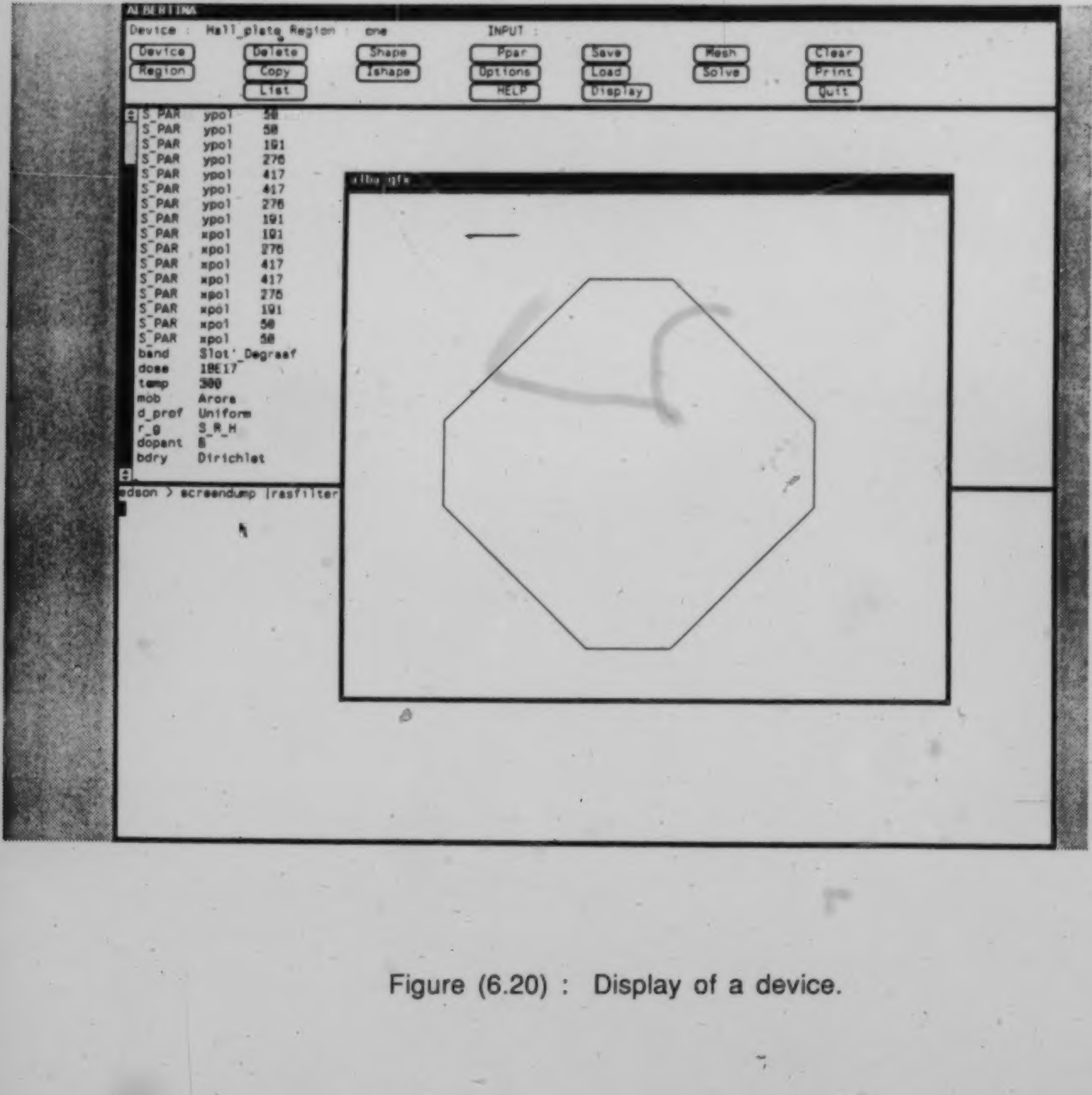
Figure (6.18) : Ppar-Mobility Menus.



```
ALBERTINA
Device :  clear      Region :              INPUT :
Region does not exist
erikson >
```

Device
Region
Delete
Copy
List
Shape
Ishape
Ppar
Options
HELP
Save
Load
Display
Geometry
Mesh
D_Profile
ES_Field
Mesh
Solve
Clear
Print
Quit

Figure (6.19) : Menu of Display Panel.

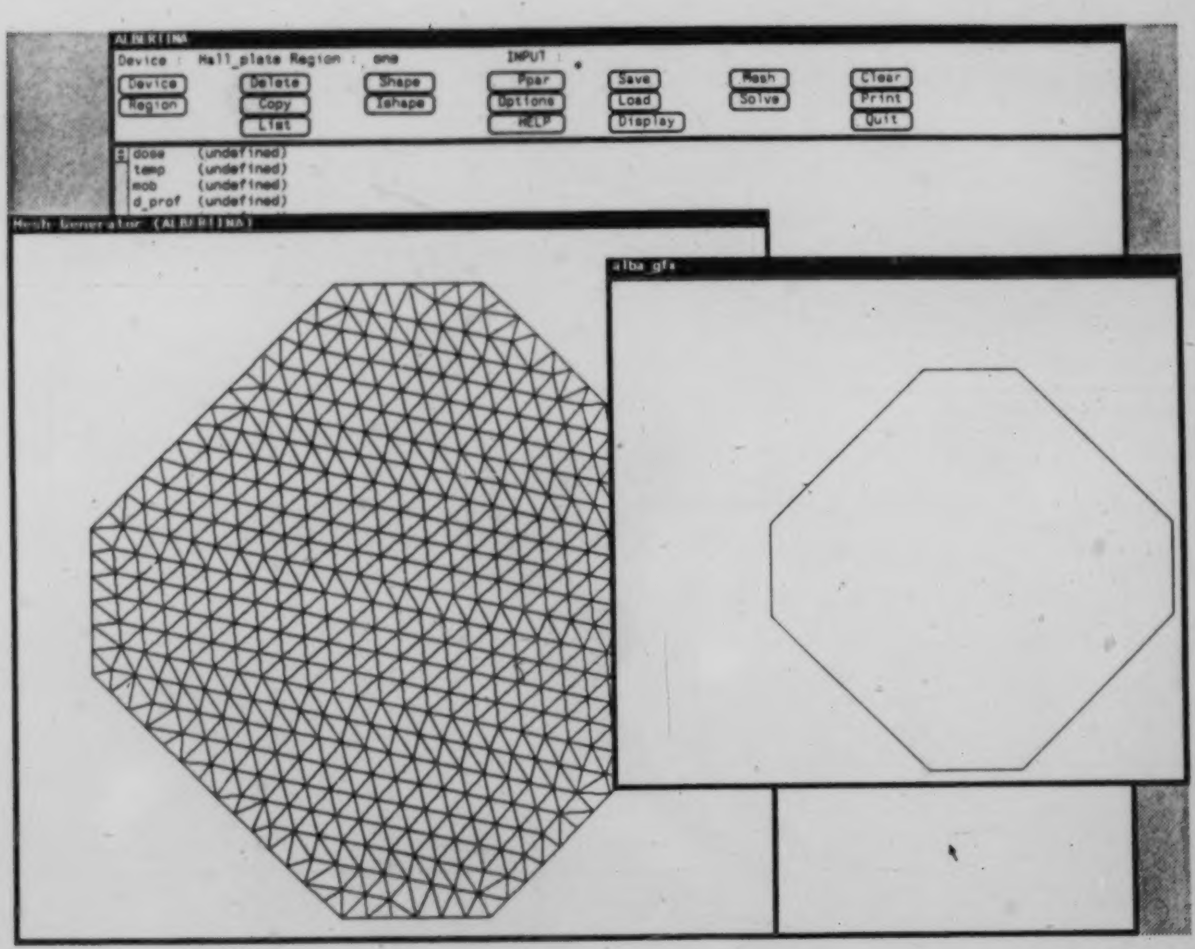

ALBERTINA
Device
Region
Delete
Copy
List
Shape
Ishape
Ppar
Options
HELP
Save
Load
Display
Mesh
Solve
Clear
Print
Quit
alba_gfx

```
Device :  Hall_plate Region :  one          INPUT :
```

```
S_PAR   ypol    50
S_PAR   ypol    50
S_PAR   ypol    101
S_PAR   ypol    276
S_PAR   ypol    417
S_PAR   ypol    417
S_PAR   ypol    276
S_PAR   ypol    101
S_PAR   xpol    101
S_PAR   xpol    276
S_PAR   xpol    417
S_PAR   xpol    417
S_PAR   xpol    276
S_PAR   xpol    101
S_PAR   xpol    50
S_PAR   xpol    50
band    Slot'_Degraaf
dose    10E17
temp    300
mob     Arora
d_prof  Uniform
r_0     S_R_H
dopant  0
bdry    Dirichlet
```

```
edson > screendump |rasfilter
```

Figure (6.20) : Display of a device.


ALBERTINA
Device
Region
Delete
Copy
List
Shape
Ishape
Ppar
Options
HELP
Save
Load
Display
Mesh
Solve
Clear
Print
Quit
Mesh Generator (ALBERTINA)
alba_gfx

Device : Hall_plate Region : one INPUT :

```
dose    (undefined)
temp    (undefined)
mob     (undefined)
d_prof  (undefined)
```

Figure (6.21) : Invoking the Mesh Generator.

7. Discussion and Outlook

In this work we have implemented a state-of-the art user interface for a semiconductor device simulation package. The interface provides an experienced user with tools and mechanisms to execute the simulation steps with ease and flexibility. The tools are used to define the geometry and physics of the simulated device. The mechanisms allow concurrent execution of various simulation steps.

The simulated device is defined in terms of regions. Each region is described and stored in dynamic and versatile tree structures. The region geometry is stored in a sub-tree that is designed in a uniform and simple structure to allow the description of various geometric shapes. The physical description of a region is encompassed in one node of the region tree description. The set of operations used to implement such descriptions are constructs of simple and well defined primitive operations. In this thesis we have provided explanations of the methods, techniques and algorithms used to implement the simple operations, defined as the "System Primitive Operations". The algorithmic constructs of these operations are offered to the user and have been defined as the "User Primitive Operations". The mechanisms offered by the user interface provides the user with means to invoke, display, inspect and modify various parameters of the simulated device. The user interface utilized the latest techniques in presenting the "User Primitive Operations". The operations are offered to the user through a graphical layer whereby all operations are displayed as invokable panels. Invoking a panel will provide the user with either a hierarchy of menus or will directly invoke a step of the simulation process. The menus are used to define the region geometry and physics. The physical definition of the device is provided from the library of "Physical Parameters", from which a user can select a model and assign it to an attribute of the physical descriptor node. The fundamental parameters of the equations modeling the behavior of semiconductor devices are included in this node. Numerical values of the physical parameters are calculated by independent programs that are invoked through a simple and versatile mechanism, where an interprocess pipe is used to create a communication channel.

Once a device has been described the user can start invoking the "External Modules" that either provide a further step towards solving the variables of the semiconductor device equations (i.e. the mesh generator and the PDE solver), or provide a display of some facet of the device through the graphics package. The modules are completely independent programs and are invoked through another mechanism where they are concurrently executed with the user interface and other processes. The user interface will monitor the data flow and apply the appropriate data conversions to meet each module's format. Errors are detected and recovered by the interface's monitoring mechanism. Monitoring insures data availability and that it contains the latest information.

The above mechanisms do not only facilitate the user interaction with the system but they also provide a flexible and expansible environment. Each module can be separately developed, modified and expanded with minor module/module interface problems. Furthermore the system expansion as a whole (to incorporate more modules, or extra models within a module) would require very little effort and time by the system administrator.

We believe that restructuring Albertina has provided a user friendly software environment with unique expansion and flexibility (i.e. maintainability) features. The implementation of such interface techniques is novel to the semiconductor device simulation area.

The user interface has been implemented on a Sun workstation in a Unix environment. Thus, it's portability to other systems is limited. On the other hand, the growing use of similar workstations with the same features and the growing trend to standardize graphics tools will help solve this problem.

The user interface and Albertina as a whole could be further enhanced by adding further process simulation steps. Adding these steps to the existing ion implantation doping profile functions would fulfill the latest requirements of simulation packages. On the other hand, a "Circuit Simulator" with parameter extraction could easily be developed or incorporated. This could be called the "horizontal expansion".

Another line of expansion is a "vertical", upward in the sense of creating new user operations of higher sophistication levels to aide a novice user. Such operations offer guidance through inference capabilities. Such an expansion will allow the package to interpret the results and suggest strategies and solutions to the user. It would also be able to estimate the validity of the current step invoked by the user. In other words such expansion would provide the package with intelligence mechanisms and heuristics to provide initial guesses and realistic values. The requirements to achieve such an expansion would include the upgrading of the current structures used in describing the simulated device. The upgrading will provide the inference mechanisms with all the dimensions describing the device within one plane. The structure upgrading could also incorporate three dimensional description of the device geometry and physical parameters. This will inherently require an upgrading of the "System Primitive Operations" and the mechanisms used in manipulating the data structures. These mechanisms could also allow the user to overlap regions and rely on the mechanisms to sort them, and to provide doping profiles that would account for the overlapped regions and return the algebraic sum of the doping concentration.

Furthermore, the models used in describing the physical parameters could be improved to incorporate other semiconductor materials e.g. GaAs and expand the list of recognizable dopants. Further sophisticated models of carrier mobility, generation and recombination, could be added in conjunction with the latest models of two dimensional doping profiles.

Finally, downward vertical expansion of Albertina would be the improvement of the basis upon which the "Basic Semiconductor Device Equations" are derived, i.e. either by solving Boltzmann's Transport Equation (BTE) to better than the first order approximation of equation (3.25), or to introduce the sub-micron equations that are based on the ballistic flight of electrons and are formulated according to the hydrodynamic model.

References

1. Zeigler, B., *Theory of Modelling and Simulation,* John Wiley and Sons, New York, 1976.

2. Townsend, P.D., Kelly, J.C., and Hartley, N.E.W., *Ion Implantation, Sputtering and their Application,* ch. 4, Academic Press, London, 1976.

3. Gummel, H.K., *A self consistent scheme for one dimensional steady state transistor calculations,* IEEE Trans. Electron Devices, VOL. ED-11, pp. 445-465, 1964.

4. Slotboom, J.W., *Iterative scheme for 1- and 2-dimensional D.C. transistor simulation,* Electron Lett., VOL. 5, pp. 677-678, 1969.

5. Mock, M.S., *A two-dimensional mathematical model of the Insulated-Gate Field Effect Transistor,* Solid State Electron., VOL. 16, pp. 601-609, 1973.

6. Selberherr, S., *Analysis and Simulation of Semiconductor Devices,* Springer-Verlag, Wien, 1984.

7. Baccarani G., Guerrieri R., Ciampolini P., and Rudan, M., *Physical Models for Numerical Device Simulation,* in : Process & Device Modeling, Volume 1, Ed. W.L. Engl, North Holland, pp. 107-158, 1986.

8. Engl. W.L, Dirks H., and Meinherzagen, H.K., *Device modeling,* Proc. IEEE, VOL. 71, pp. 10-33, 1983.

9. Bank, R., Rose, D., and Fichtner, W., *Numerical methods for semiconductor device simulation,* IEEE Trans. Electron Devices, VOL. ED-30, pp. 1018-1041, 1983.

10. Fichtner, W., Rose, D., and Bank. R., *Semiconductor device simulation,* SIAM. J.

Sci. Stat. Comput., VOL. 4, pp. 391-415, 1983.

11. Thacker, W.C., *A brief review of techniques for generating irregular computational grids,* International Journal for Numerical Methods in Engineering, Vol. 15, pp. 1335-1341, 1980.

12. Simpson, R.B., *A survey of two dimensional finite element mesh generation,* Proc. 9th Manitoba Conference on Numerical Math. and Computing, pp. 49-124, 1979.

13. *The Interface and Integration of Process Device and Circuit Models,* NASCODE V Conference, Trinity College, Dublin, Ireland. Ed. J.J. Miller, Boole Press, 1987.

14. Blue, J., and Wilson, C., *Two-Dimensional analysis of semiconductor devices using general-purpose interactive PDE software,* IEEE Trans. Electron Devices, VOL. ED-30, pp. 1056-1064, 1983.

15. Rudan M., Guerrieri R., Ciampolini P., and Baccarani G., *Discretization Strategies and Software Implementation for 2D_Device Simulator,* in : New Problems and Solutions for Device and Process Modeling., Ed. J.J. Miller, Boole Press, Dublin, Ireland, pp.110-121, 1986.

16. Baccarani G., Guerrieri R., Ciampolini P., and Rudan, M., *HFIELDS : A highly flexible 2-D semiconductor-device analysis program.* Proc. NASCODE IV, pp. 3-12, Dublin, Ireland, 1985.

17. Belhadad K., Corbex, C., and Poncet, A. *Finite element and coupled continuation method for device simulation,* Proc. NASCODE IV, pp. 193-198, Dublin, Ireland, 1985.

18. Poncet, A., Corbex, C., and Belhadad K., *General Purpose Data Structures and Software Tools for Process-Device Simulator Integration.* in: The Interfaces and Integration of Process, Device and Circuit Models - An Introduction, Ed. J.J. Miller, Boole Press, Dublin Ireland, 1987.

19. Prendergast E.J., *An integrated approach to modeling*, Proc. NASCODE IV, pp. 83-97, Dublin, Ireland, 1985.

20. Engl, W.L., Laur, R., and Dirks, H.K., *MEDUSA - A simulator for Modular Circuits*, IEEE Trans. CAD, VOL. CAD_1, No. 2, pp. 85-93, 1982.

21. Hamilton, A., and Amaratunge, G., *A method for interfacing process and device simulators and results obtained from interfacing to MINIMOS*, Proc. NASCODE IV, pp. 3-12, Dublin, Ireland, 1985.

22. Lombardi C., Vanzi, M., and Torri, E., *IDAS : An interactive device analysis environment*, Proc. NASCODE IV, pp. 384-389, Dublin, Ireland, 1985.

23. Selberherr, S., Shutz, A., and Potzl, H.W., *MINIMOS, A two dimensional MOS transistor analyzer*, IEEE Trans. Electron Devices, VOL. ED 27, pp. 1540-1550, 1980.

24. Selberherr, S., *The status of MINIMOS*, in : Simulation of Semiconductor Devices and Processes, Vol. 2. Proc. International Conference on Simulation of Devices and Processes, pp. 2-15, Ed: Board, K., and Owen D.R. J., Pineridge Press, Swansea, U.K., 1986.

25. Bevan, N., *The Design of User-Friendly Systems for Generating Intelligent Dialogues*, in : Integrated Interactive Computing Systems, Ed.: Degano, P., and Sandwall, E., North Holland Publishing Company, Amsterdam, pp. 333-342, 1983.

26. Enderle, G., *Report on the Interface of UIMS to the Application,* in : User Interface Management Systems, Ed. Pfaff, G., pp. 21-29, Springer-Verlag, Wien, 1983.

27. Smith, J.J., *SUSI - A Smart User-System Interface,* in : People and Computers : Designing the Interface. Proc. of the British Computer Society, Human Interaction Specialist Group, University of East Anglia, 1985. Ed: Johnson, P., and Cook, S., Cambridge University Press, London, pp. 211-230, 1985.

28. Hanson, S.J., *Interface Design and Multivariate Analysis of Unix Command Use,* ACM Trans. of Office Information Systems 2. No. 1, 1984.

29. Newman, W., Stephens, N., and Sweetman, P., *A Window Manager with a Modular User Interface,* in : People and Computers : Designing the Interface. Proc. of the British Computer Society, Human Interaction Specialist Group, University of East Anglia, 1985. Ed: Johnson, P., and Cook, S., Cambridge University Press, London, pp. 415-426, 1985.

30. Reichman, R., *Communication Paradigms for a Window System,* in : User Centered System Design, Ed : Norman, D., and Draper, S., Lawrence Erlbaum Ass. Publishers, Hillsdale, New Jersey, pp. 285-313, 1986.

31. Fischer, G., and Boecker, H., *The Nature of Design Processes and how Computer Systems can Support them,* in : Integrated Interactive Computing Systems, Ed.: Degano, P., and Sandwall, E., North Holland Publishing Company, Amsterdam, pp. 73-86, 1983.

32. Seeger, K., *Semiconductor Physics, An Introduction,* Springer-Verlag, New York, 3rd Edition, 1985.

33. Ryssel, H., Haberger, K., Hoffman, K., Prinke, G., Dumcke, R., and Sachs, A., *Simulation of doping processes,* IEEE Trans. Electron Devices, Vol. ED-30, pp.

1484-1492, 1980.

34. Sze, S. M., *Physics of Semiconductor Devices*, J. Wiley and Sons, Second Edition, New York, 1981.

35. Seidel, T.E., *Ion Implantation*, in : VLSI Technology, Ed: S.M. Sze, McGraw-Hill, New York, pp. 219-264, 1983.

36. Selberherr, S., Guerrero, E., *Simple and accurate representation of implantation parameters by low order polynomials*, Solid State Electronics, Vol. 24, pp. 591-593, 1981.

37. Hofker, W.K., *Concentration Profiles of Boron Implantations in Amorphous and Polycrystalline Silicon*, Philips Research Report, Vol. Supp. # 8, pp 41-57, 1975.

38. Furukawa, S., Matsumura, H., and Ishiwara, H., *Theoretical considerations on lateral spread of implanted ions*, Japanese Journal of Applied Physics, Vol. 11, No. 2, pp. 134-142, 1972.

39. Lindhard, J., Schraff, M., and Schiφtt, H., *Range concepts and heavy ion ranges*, Mat. Fys. Medd. K. Dan. Vidensk. Selsk., Vol. 33, No. 14, pp.1-42, 1963.

40. Hobler, G., Langer E., and Selberherr, S., *Two Dimensional Modeling of Ion Implantation*, in: Simulation of Semiconductor Devices and Processes, Vol. 2, Ed. Board, K., and Owen, D., Pineridge Press, Swansea, U.K., pp. 256-270, 1986.

41. Giles, M., *Calculation of Ion Implantation Profiles for Two Dimensional Process Modeling*, in : Simulation of Semiconductor Devices and Processes, Vol. 2, Ed. Board, K., and Owen, D., Pineridge Press, Swansea, U.K, pp. 233-246, 1986.

42. Smith, R.A., *Semiconductors*, Second Edition, Cambridge University Press, Cam-

bridge, 1979.

43. Conwell, E., and Weisskopf, V., *Theory of impurity scattering in semiconductors*, Physical Review, VOL. 77, pp. 388-390, 1950.

44. Brooks, H., *Scattering by ionized impurities in semiconductors*, Physical Review, VOL. 83, p. 879, 1951.

45. Caughey, D.M., and Thomas, R.E., *Carrier mobilities in silicon empirically related to doping profile*, Proc. IEEE, VOL. 55, pp. 2192-2193, 1967.

46. Scharfetter, D.L., and Gummel, H.K., *Large signal analysis of silicon read diode oscillator*, IEEE Trans. Electron Devices, VOL. ED-16, pp. 64-77, 1969.

47. Thornber, K.K., *Applications of scaling to problems in high-field electronic transport*, J. Appl. Phys. , VOL. 52, pp. 279-290, 1981.

48. Canali, C., Majni, G., Minder, R., and Ottaviani, G., *Electron and hole drift velocity measurements in silicon and their empirical relation to electric field and temperature*, IEEE Trans. Electron Devices, ED-22, pp. 1045-1047, 1975.

49. Arora, N.D., Hauser, J.R., and Roulston, D.J., *Electron and hole mobilities in silicon as a function of concentration and temperature*, IEEE Trans. Electron Devices, VOL. ED-29, pp. 292-295, 1982.

50. Yamaguchi, K., *A mobility model for carriers in the MOS inversion layer*, IEEE Trans. Electron Devices, VOL. ED-30, pp. 658-663, 1983.

51. Hall, R.N., *Electron-hole recombination in germanium*, Physical Review, Vol. 87, p. 387, 1952.

52. Shockley, W., and Read, W. T., *Statistics of recombination of holes and electrons,* Physical Review, Vol. 87, pp. 835-842, 1952.

53. Fossum, J. G., *Computer aided numerical analysis of silicon solar cells,* Solid State Electron., VOL. 19, pp. 269-277, 1976.

54. Fossum, J.G., and Lee, D.S., *A physical model for the dependence of carrier lifetime on doping density in nondegenerate silicon,* Solid State Electron., VOL. 25, pp. 741-747, 1982.

55. Engl, W.L, and Dirks, H.K, *Models of Physical Parameters,* in : An introduction to the Numerical Analysis of Semiconductor Devices and Integrated Circuits, Ed: J.J. Miller, Boole Press, Dublin, pp. 42-46, 1981.

56. Okuto, Y., and Crowell, C.R., *Threshold energy effect on avalanche breakdown voltage in semiconductor junctions,* Solid State Electron., VOL. 18, pp. 161-168, 1975.

57. Slotboom, J.W., and De Graaf, H.C., *Measurements of bandgap narrowing in silicon bipolar transistor,* Solid State Electron., VOL. 19, pp. 857-862, 1976.

58. Van Overstraeten, R., DeMan H.J., and Mertens, R., *Transport equation in Heavy-doped silicon,* IEEE Trans. Electron Devices, VOL. ED-20, pp. 290-298, 1973.

59. Thurmond, C.D., *The standard thermodynamic function of the formation of electrons and holes in Ge, Si, GaAs and GaP,* J. Electrochem. Soc., VOL. 122, pp. 1133, 1975.

60. Joerg, W., Nathan, A., and Saad, R., *FlexiFace : An interactive package to provide a consistent & extensible user interface for a scattered device modelling*

system, Proc. IASTED-ASM 1987, pp. 53-59, Santa Barbara, California.

61. Kruse, R., *Data Structures and Program Design*, Prentice-Hall Inc., New Jersey, 1984.

62. Kernighan, B.W., and Ritchie, R., *The C programming language*, Prentice Hall Inc., New Jersey, 1978.

63. *Sunview System Programmer's Manual*, Sun Microsystems Inc., Revision-B, 1986.

64. *Sunview programmer's Guide*, Sun Microsystems, Inc. Revision-B, 1986.

65. ANSI X3H3 84/85. *Information Processing Computer Graphics Virtual Device Interface (VDI) Functional Description*, March 1984.

66. *SunCGI Reference Manual*, Sun Microsystems, Inc. Revision-B, 1986.

67. *Status Report of the Graphics Standards Planning Committee.* Computer Graphics. Vol. 13, No. 3, 1979.

68. Foley, J.D., and Van Dam., A. *Fundamentals of Interactive Computer Graphics* Addison-Wesley, Reading Massachusetts, 1982.

69. *Unix Programmer's Manual*, version 4.3 BSD, 1985.

70. Kernighan, B.W., and Pike, R., *The Unix Programming Environment*, Prentice Hall Software Series, Prentice Hall Inc., New Jersey, 1984.

71. Joe, B., and Simpson R.B, *Triangular meshes for regions of complicated shape*, International Journal For Numerical Methods in Engineering, Vol. 23, pp. 751-778, 1986.

Appendix I : "C" Definitions of Nodes

1. Geometry Descriptor :

The Geometry Descriptor node is defined as:

```
typedef struct Gpar{
        char *value;      /* Pointer to the node’s value */
        char *type;       /* Pointer to the node’s type*/
        struct Gpar *left; /*Link to the node’s left child */
        struct Gpar *right;        /* Link to the node’s right child */
}Gpar;
```

2. Physical Parameter Descriptor :

The Physical Parameter Descriptor node is defined as:

```
typedef struct Ppar{
          char *band;   /*pointer to band gap narrowing model*/
          char *conc;   /*Pointer to dopant dose*/
          char *temp;   /*Temperature */
          char *mob;    /*Pointer to mobility model*/
          char *d_prof; /*Pointer to doping profile model */
          char *r_g;    /*Pointer to Generation and recombination model*/
          char *dopant; /*Pointer to dopant species */
          char *bdry;   /* Pointer to boundary condition */
          struct Ppar *pext;
     }Ppar;
```

3. Region Descriptor :

The structure Region descriptor node :

```
typedef struct Region{
     char *name;;        /* Pointer region's name */
     char *value;        /* Pointer to region's value */
     struct Gpar *g_link; /* Pointer to geometry subtree*/
     struct Ppar *p_link; /*Pointer to Physical descriptor node */
     struct Region *rext; /*Pointer to next region */
} Region;
```

Appendix II : Constants of Physical Models

Carrier	$\mu_{\min}\ \frac{cm^2}{V.s}$	$\mu_{\max}\ \frac{cm^2}{V.cm}$	α_c	$N_{ref}\ cm^{-3}$
Electron	65	1330	0.72	8.5×10^{16}
Hole	47.20	495	0.78	6.3×10^{16}

Table (II.1) : Caughey-Thomas mobility model (eqn. 3.71).

Carrier	$E_c\ \frac{V}{cm}$	β_c
Electron	8.0×10^{3}	2
Hole	1.95×10^{4}	1

Table (II.2) : Caughey-Thomas mobility model (eqn. 3.72).

Carrier	$u_l\ \frac{cm}{sec}$	G
Electron	4.9×10^{6}	8.8
Hole	4.9×10^{6}	1.60

Table (II.3) : Thornber-Gummel mobility model (eqn. 3.74).

Carrier	$\mu_o \frac{cm^2}{V.sec}$	$N_{ref}\ cm^{-3}$	S
Electron	1330	3.0×10^{16}	350
Hole	495	4.0×10^{16}	81

Table (II.4) : Scharfetter-Gummel mobility model (eqn. 3.77).

Carrier	$\mu_0 \frac{cm^2}{V.s}$	S	N cm^{-3}	A V/cm	F	B V/cm
Electron	1440	350	3.0×10^{16}	3.5×10^3	8.8	7.4×10^3
Hole	480	81	4.0×10^{16}	6.1×10^3	1.6	2.5×10^4

Table (II.5) Scharfetter-Gummel mobility model (eqn. 3.78).

Carrier	a $\frac{cm^2}{Vs}$	b $\frac{cm^2}{Vs}$	c cm^{-3}	α_a	β_a	γ_a
Electron	88	1252	1.432×10^{17}	-0.57	-2.23	2.546
Hole	54.30	407	2.67×10^{16}	-0.57	-2.23	2.546

Table (II.6): Arora's mobility model (eqn 3.79).

Carrier	$\alpha_y \frac{cm}{V}$
Electron	1.54×10^{-5}
Hole	5.35×10^{-5}

Table (II.7) : Yamaguchi mobility model (eqn 3.81).

Carrier	x^0 cm	$E^{crit} \frac{V}{cm}$	$E^t_{cirt} \frac{V}{cm}$
Electron	5.0×10^{-7}	1.0×10^{4}	1.80×10^{5}
Hole	4.0×10^{-7}	8.0×10^{3}	3.80×10^{5}

Table (II.8) : Selberherr mobility model (eqns. 3.82 - 3.85).

τ_0^n s	τ_o^p s	N_o^n $cm-3$	N_o^p $cm-3$
3.95×10^{-4}	3.52×10^{-5}	7.1×10^{15}	7.1×10^{15}

Table (II.9) : SRH model (eqn. 3.86, 3.87).

Appendix III : Illustration of Results.

In the following pages of this appendix we present plots of the projected range R_p and standard deviation ΔR_p for the four dopant species considered in Albertina. The calculations are performed according to equations (3.46) and (3.47), and are interactively calculated as by the subroutine *proj_dev* used in the algorithm constructing the dop PPSC (section 6.5.3).

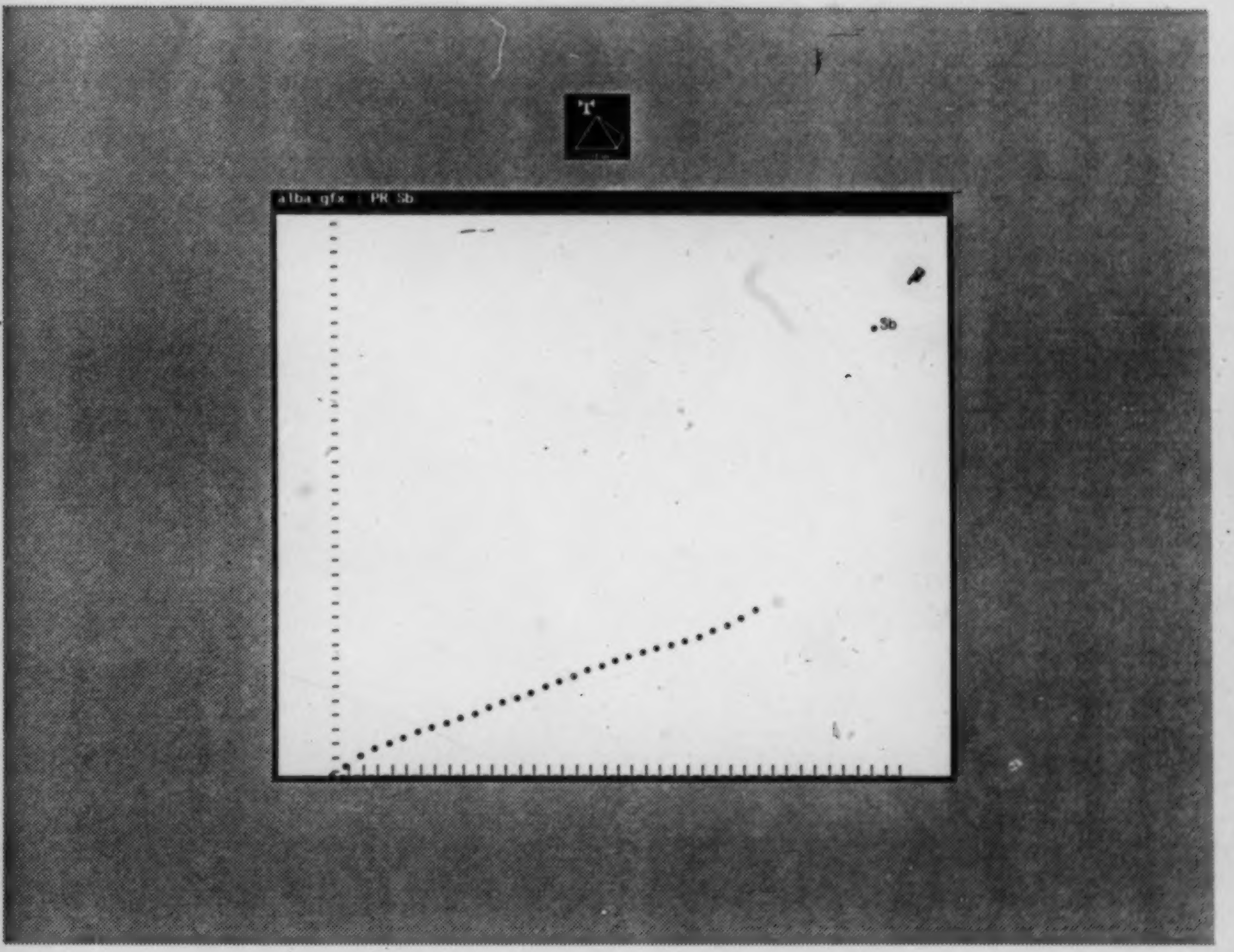

alba_gfx : PR Sb
Sb

Figure (III.1) : Projected range of antimony in silicon.

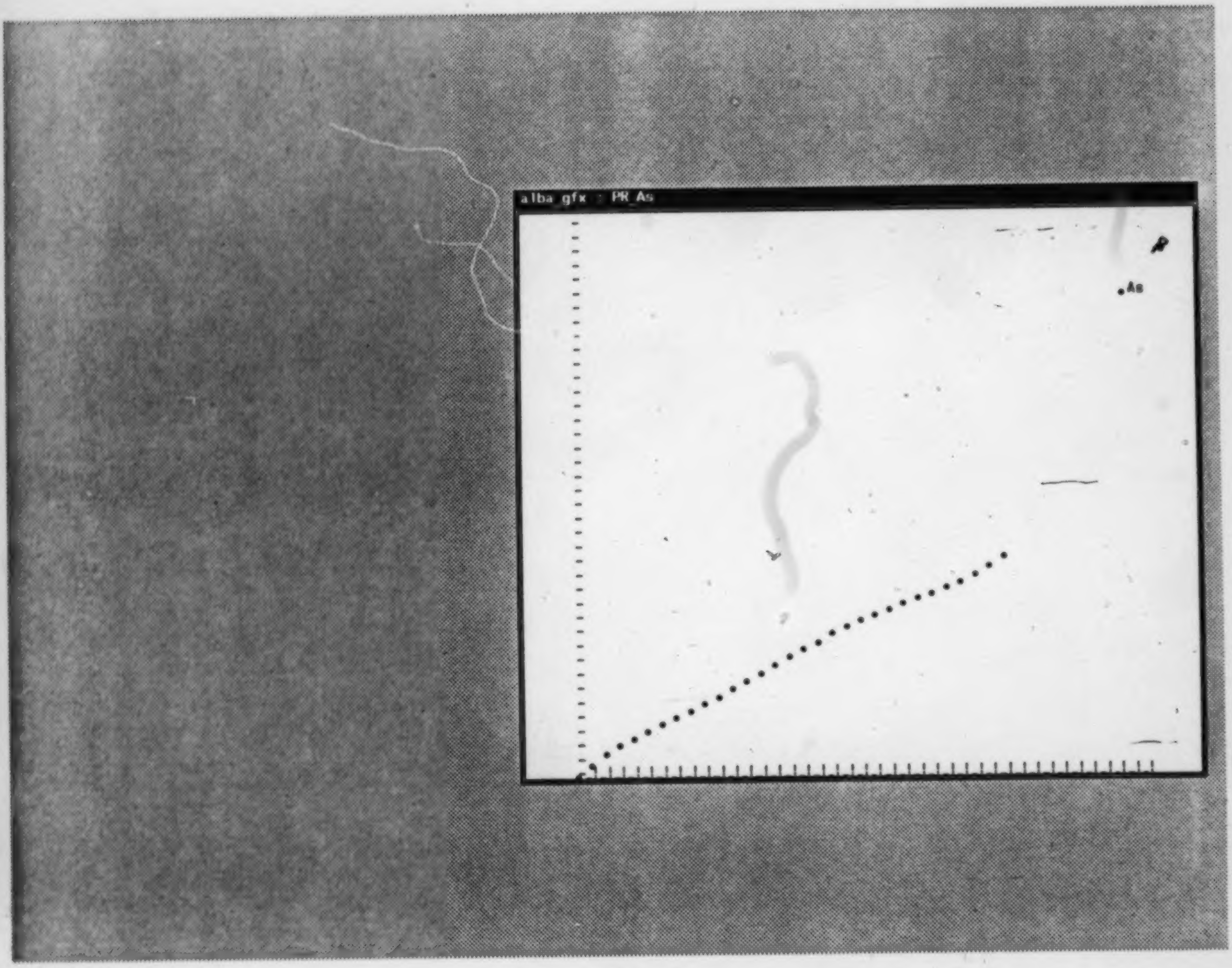


Figure (III.2) : Projected range of arsenic in silicon.

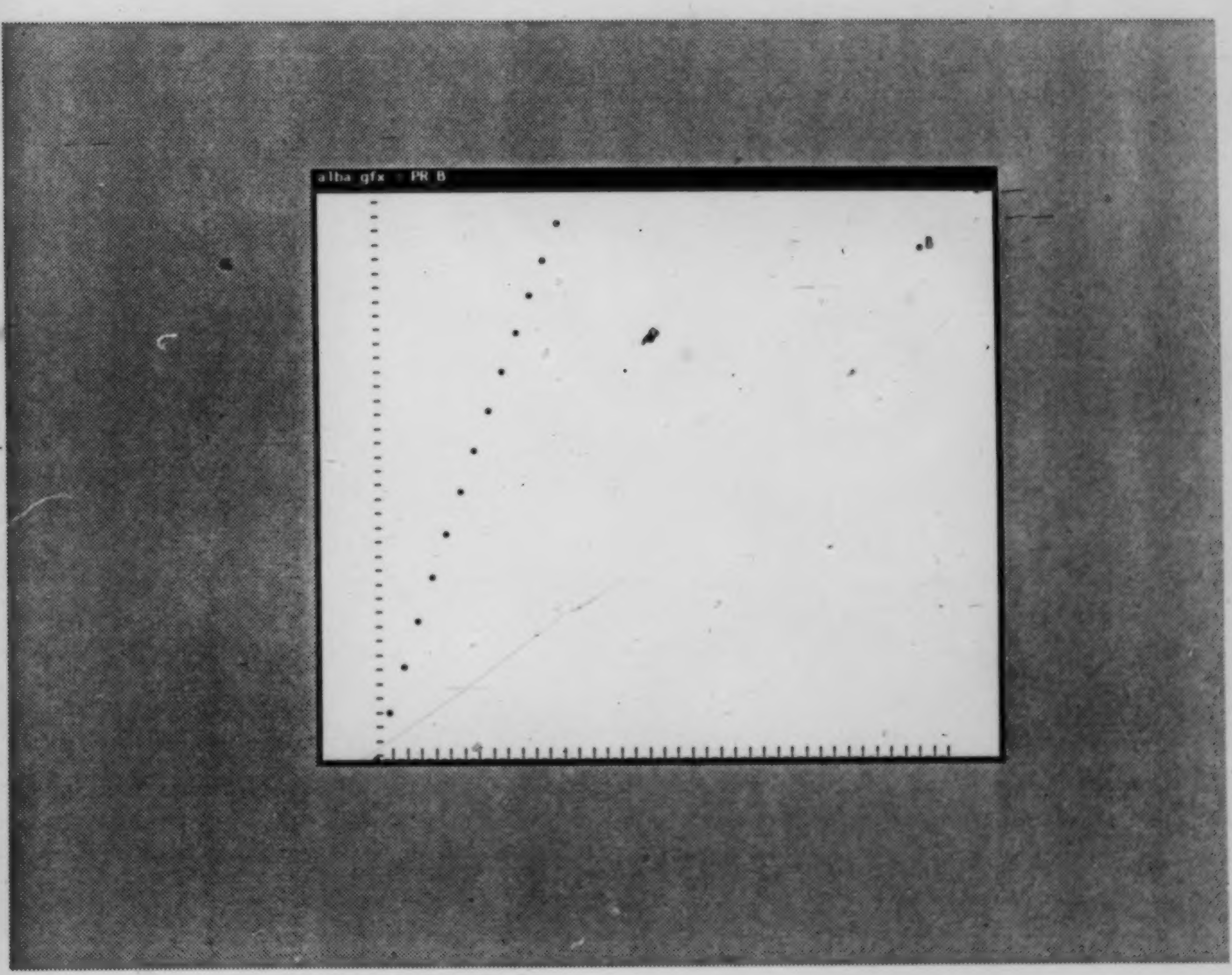


Figure (III.3) : Projected range of boron in silicon.

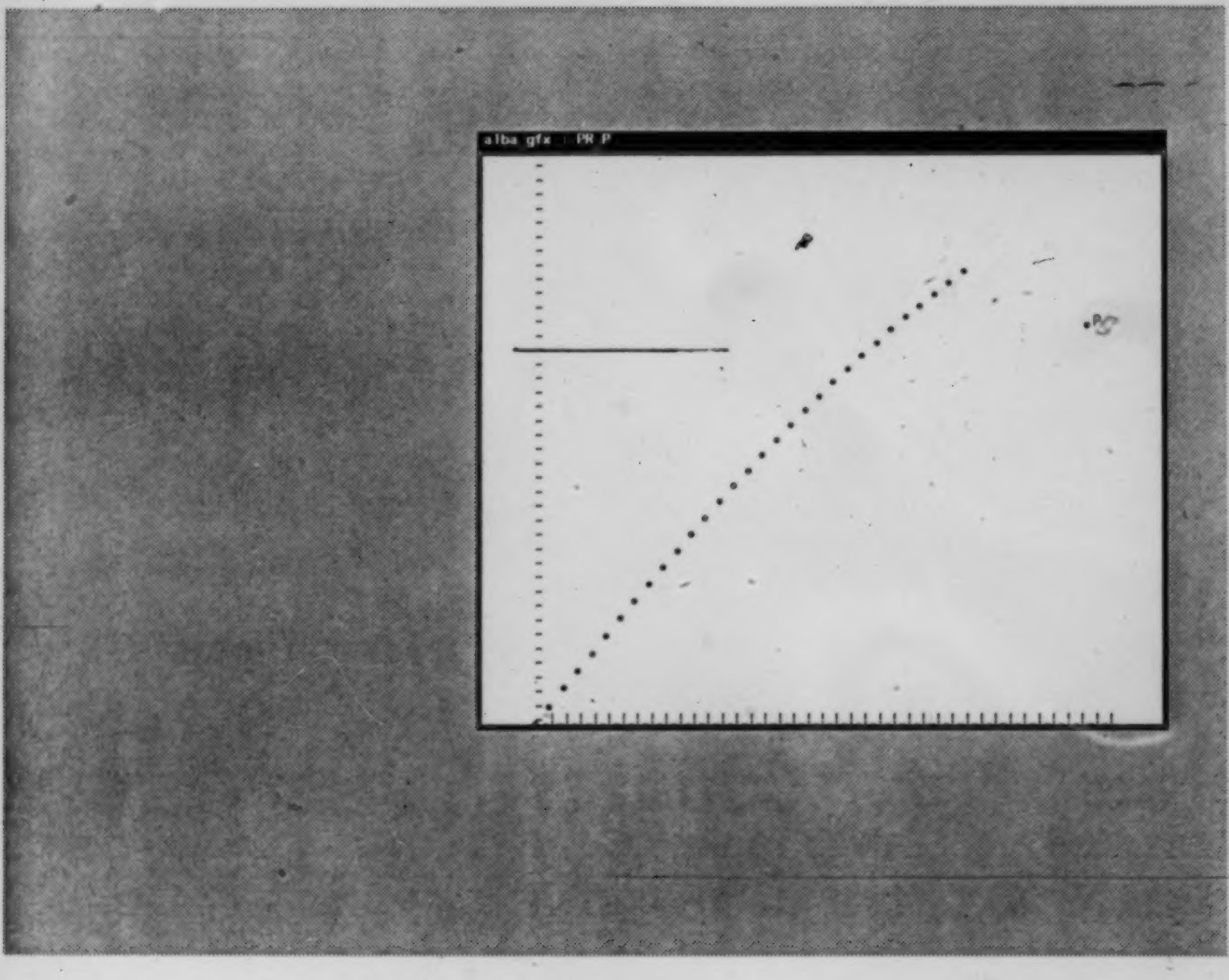


Figure (III.4) : Projected range of phosphorus in silicon.

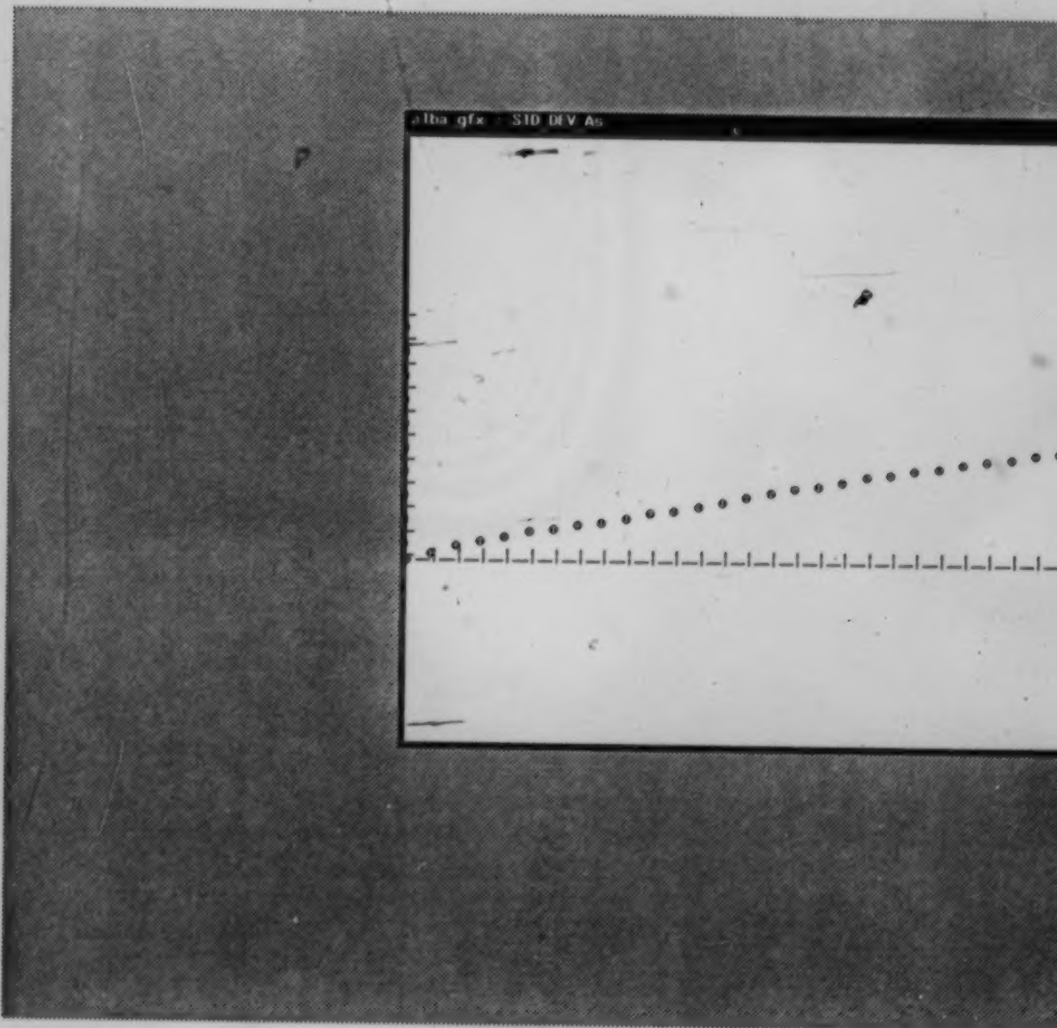


Figure (III.5) : Standard deviation of antimony in sili

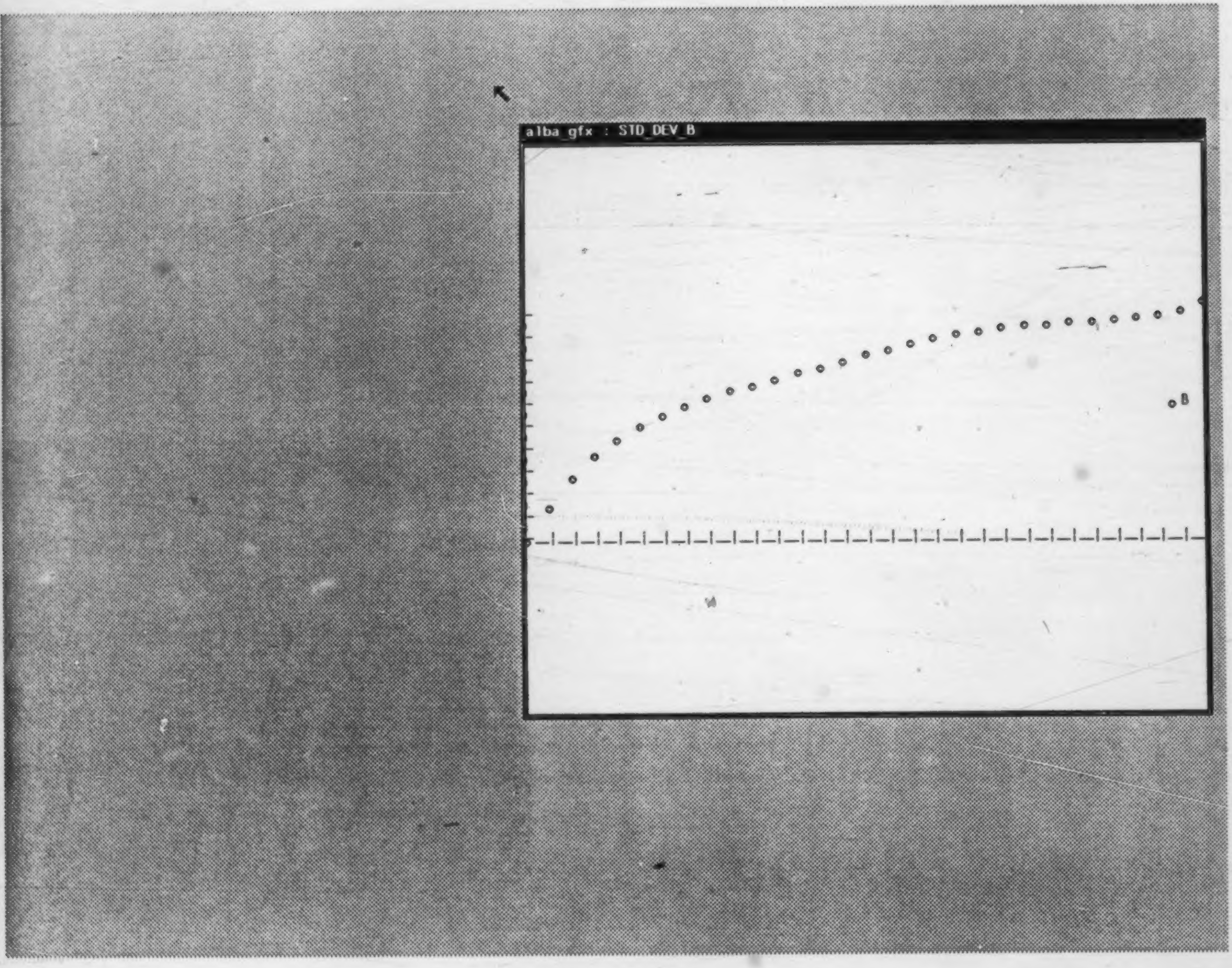


Figure (III.6) : Standard deviation of boron in silicon.

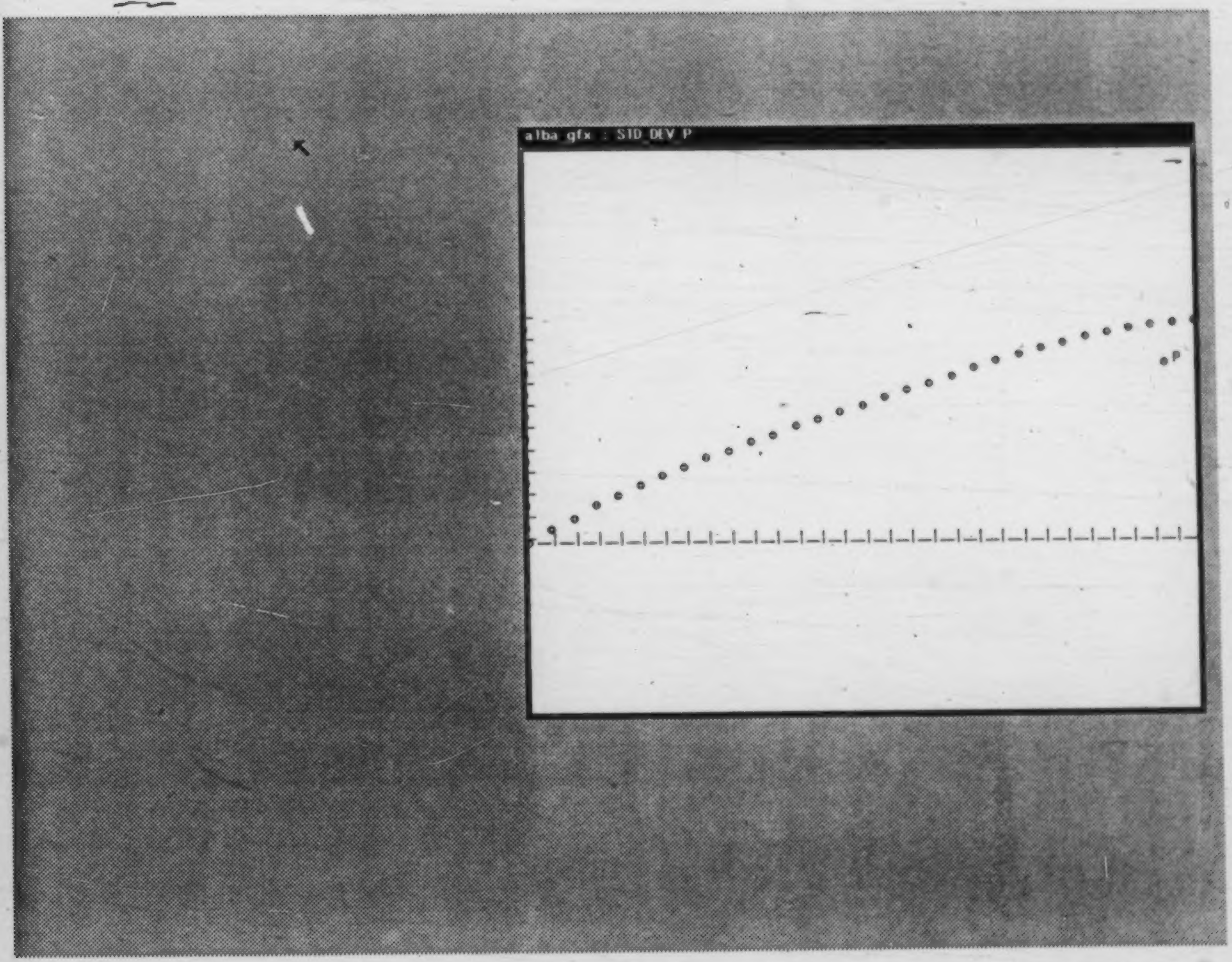


Figure (III,7) : Standard deviation of phosphorus in-silicon.